

A linguagem PASCAL conhece um êxito crescente pelas suas interessantes qualidades em relação à necessidade da programação estruturada e também pela possibilidade de escrever diferentes compiladores nesta linguagem, assim como pela sua definição formal. Estas qualidades permitem utilizar a PASCAL em todos os domínios e, em particular, no campo do ensino. Neste manual, são analisadas todas as características da linguagem, em capítulos separados, subdividindo-se estes em partes relativas à utilização, à sintaxe, à semântica e ainda ao modo de utilização de cada característica. Vocabulário, operações, ordinais, caracteres, quadros, registos, constantes e variáveis, e ainda os diversos tipos de instruções, são algumas das rubricas incluídas neste utilíssimo guia prático.



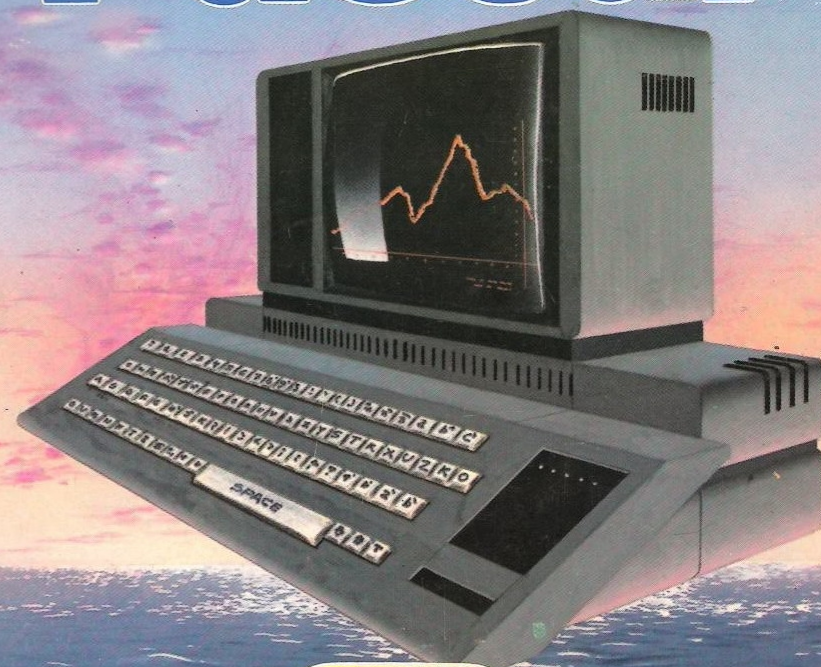
EDITORIAL PRESENÇA

Guia de Pascal

176

DAVID WATT

GUIA DE Pascal



TEMPOS
LIVRES

CULTURA E TEMPOS LIVRES

1. ABC do Xadrez, *Peter Trifunovitch e Sava Vulkovitch*
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 Arquitetura 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 Transistores, *Fernand Huré*
19. Karaté I, *Albrecht Pflüger*
20. Iniciação ao Radiocomando dos Modelos Reduzidos, *C. Péricon*
21. Construa o seu Receptor, *B. Fighiera*
22. Montagens Electrónicas, *B. Fighiera*
23. O Berbequim Eléctrico, *Villy Dreier*
24. Cactos, *J. Nilas Jensen*
25. Iniciação à Alta Fidelidade, *Peter Turner*
26. O Aquário de Água Doce, *Paulo de Oliveira*
27. ABC do Ténis, *Fonseca Vaz*
28. Karaté II, *Albrecht Pflüger*
29. ABC da Criação de Canários, *Curt Af Enehjelm*
30. Ginástica Feminina, *Sonja Helmer Jensen*
31. Cartomancia, *Rhea Koch*
32. Calculadoras Electrónicas de Bolso, *E. Dam Ravn*
33. O Pastor Alemão, *Gilles Legrand*
34. Xadrez – Teoria do Meio Jogo I, *I. Bondarevsky*
35. Manual do Super 8 – I, *Myron A. Matzkin*
36. ABC da Criação de Piriquitos, *Cyril H. Rogers*
37. O Livro dos Gatos, *Barber Gerber e Horst Bielfeld*
38. Manual do Super 8 – II, *Myron A. Matzkin*
39. ABC do Mergulho Desportivo, *Walter Mattes*
40. Circuitos Integrados/Aplicações Práticas, *F. Bergtold*
41. A Apicultura, *H. R. C. Riches*
42. ABC do Cultivo das Plantas, *H. G. Witham Fogg*
43. ABC da Criação de Pombos, *Kai R. Dahl*
44. Construção de Caixas Acústicas de Alta Fidelidade, *R. Brault*
45. Raças de Canários, *Klaus Speicher*
46. Jogos de Cartas, *Graciano Dolma*
47. Cocker Spaniels, *H. S. Lloyd*
48. ABC da Caça, *Fabian Abril*
49. Aprenda Televisão, *Gordon J. King*
50. Iniciação à Pesca, *Juan Nadal*
51. Basquetebol, *Marius Norregard*
52. Cães de Caça, *Santiago Pons*
53. Aprenda Electrónica, *T. L. Squires e C. M. Deason*
54. A Avicultura, *Jim Worthington*
55. A Produção de Coelho, *P. Surdeau e R. Henaff*
56. ABC dos Computadores, *T. F. Fry*
57. Natação para Crianças, *John Idorn*
58. O Boxer, *Anni Mortensen*
59. Voleibol, *Ole Hansen e Per-Göran Persson*
60. Iniciação à Vela, *Donald Law*
61. ABC da Filatelia, *Jacqueline Caurat*
62. A Pesca à Beira-Mar, *J.-M. Boëlle e D. Doyen*
63. Enxerto de 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, *Luis 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, *Matias Josa*
79. Rádio – Fundamentos e Técnicas, *Gordon J. King*
80. Como Fazer Gelados, *Sylvie Thiébaut*
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. Hasselfeldt*
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 Coelho, *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, *Jérôme Sauze*
106. Crepes – Doces e Salgados, *Florence Arzel*
107. Aperitivos e Entradas, *Myrette Tiano*
108. Ténis de Mesa, *Leslie Woollard*
109. Aprenda Surf, *R. Abbott e M. Baker*
110. Futebol – Técnica e Tática, *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-W. J. Weber*
117. Gansos e Patos, *Marie Mourthe*
118. Iniciação ao Kung-Fu, *A. P. Harrington*
119. Electrónica e Fotografia, *Hans-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, *T. M. Normand*
129. Sumos e Batidos, *Manfred Donderski*

130. Introdução à Programação de Microcomputadores, *Peter C. Sanderson*
131. Aprenda Croché, *Uta Mix*
132. ABC do Microprocessador, *P. Mélusson*
133. Guia Prático de Basic, *Roger Hunt*
134. Introdução à Electrónica Digital, *Ian R. Sinclair*
135. ABC do Vídeo, *David Matthewson*
136. Fotografia em Movimento, *Don Morley*
137. Guia de Cobol, *Ray Welland*
138. Fotografia a Pequena Distância, *Sidney F. Ray*
139. Guia Moderno da Canaricultura, *Manuel Gonçalves*
140. Minielectrónica para Amadores, *Heinz Richter*
141. ABC da Programação de Computadores, *John Shelley*
142. Tarot - O Futuro Pelas Cartas, *Edwin J. Nigg*
143. ABC da Equitação, *Dorothy Johnson*
144. Como Programar o seu ZX 81, *Patrick Gueulle*
145. 100 Avarias TV e a Maneira Prática de as Detectar, *P. Duranton*
146. ABC da Horticultura, *Louis Giordano*
147. Basic Para Microcomputadores, *A. P. Stephenson*
148. Como Programar o seu ZX Spectrum, *Tim Hartnell e Dilwyn Jones*
149. Iniciação aos Motores Diesel, *David S. Maclean*
150. 60 Jogos Para o ZX Spectrum, *David Harwood*
151. As Linhas da Mão, *Rose Hubert*
152. Cozinha Italiana, *Rotraud Degner*
153. Manual do ZX Spectrum, *R. J. Simpson e T. J. Terrell*
154. Z80 Assembler Para o ZX Spectrum - Iniciação ao Código de Máquina, *João Paulo Fragoso*
155. Aeróbica, *Hans Schulz*
156. ABC do Atletismo, *Denis Watts*
157. 26 Programas Basic Para Microcomputadores, *Derrick Daines*
158. Aprenda Pascal no Seu Microcomputador, *Jeremy Ruston*
159. Guia Moderno da Suinicultura, *Colin Whitemore*
160. O Bar em Sua Casa - 888 Cocktails, *Aladar von Wesendonk*
161. Código de Máquina Para Principiantes, *James Walsh*
162. Código de Máquina Para Programadores Avançados, *Paul Holmes*
163. ABC da Fruticultura, *Henri Gosselin*
164. ABC da Canoagem, *Alan Hyde*
165. Guia de Fortran, *Philip Ridler*
166. Manual da Secretária, *Phillippa Ramage*
167. ABC das Antenas, *Gordon J. King*
168. Programar Aventuras no Seu Computador, *Andrew Nelson*
169. Guia do Sinclair QL, *Boris Allan*
170. Novas Aventuras no Seu ZX Spectrum, *Peter Shaw e James Mortleman*
171. O Computador no Escritório, *John Shelley*
172. Sobremesas, *Fred Timber*
173. Rádio - Do Circuito Oscilante ao Receptor de Ondas Curtas, *Richard Zierl*
174. ABC das Aberturas, *V. N. Panov*
175. A Construção de Pequenos Transformadores, *M. Dourian e F. Juster*
176. Guia de Pascal, *David Watt*

GUIA DE PASCAL

COMO USAR ESTE GUIA DE BOLSO

Cada característica da linguagem Pascal é analisada num capítulo à parte. A maior parte destes capítulos contém as seguintes partes. *Utilização* descreve o modo de emprego da característica estudada em programação e a sua importância para a linguagem. *Sintaxe* apresenta as regras gramaticais da característica estudada – regras que serão verificadas pelo compilador. *Semântica* descreve o significado da característica estudada em termos de execução do programa. *Exemplos* ilustra o modo de utilizar a característica estudada.

Cada um dos capítulos relativos a tipos de dados inclui igualmente uma secção *Operações*, que resume todas as operações que podem ser executadas sobre o tipo de dados escritos.

Notação

Os símbolos Pascal são apresentados assim: *begin* (início) *end* (fim) ; : =.

= significa “é definido como”.

| separa alternativas, isto é, “letra | algarismo” significa “letra *ou* algarismo”.

(...) é usado para agrupar alternativas.

Título original:
POCKET GUIDE TO PASCAL
© Copyright by David Watt, 1982
Tradução de Conceição Jardim e Miguel Nogueira
Capa de António Marques

Reservados todos os direitos
para a língua portuguesa à
EDITORIAL PRESENÇA, LDA.
Rua Augusto Gil, 35-A – 1000 Lisboa

[...] indica uma construção de opção, por exemplo “[*else* instrução]” indica a escolha de “*else*” seguida de uma instrução, *ou* nada.

{ ...*... } indica uma sequência de construções semelhantes, por exemplo:

(a) “{identificador*,}” indica uma sequência de identificadores separados por “,”;

(b) “{algarismo*}” indica uma sequência de um ou mais algarismos;

(c) “{*(letra | algarismo) }” indica uma sequência de zero ou mais letras ou algarismos.

2

VOCABULÁRIO: SEPARADORES

Ao nível léxico, o texto de um programa é composto por termos, fixos (*tokens*), ou seja, símbolos que determinam a estrutura sintática e o significado do programa, e *separadores*, cuja função principal consiste em facilitar a compreensão pelo utilizador humano.

Sintaxe

programa-texto = { * { * separador } termo }
separador = em branco | fim de linha | comentário
comentário = { comentário-texto }

Os separadores não têm qualquer efeito no significado de um programa. Os espaços em branco e o final de linhas podem ser usados para melhorar a apresentação do texto do programa. Os comentários podem ser usados para explicar o programa a um leitor humano. Um comentário pode incluir quaisquer caracteres e finais de linha, mas não deve incluir “,” nem “*”).

É necessário introduzir pelo menos um separador entre dois termos fixos consecutivos que sejam palavra-chave, identificadores, constantes inteiras sem sinal ou constantes reais desprovidas de sinal. Não se devem introduzir separadores no interior de um termo fixo.

Os separadores são ignorados nas regras sintáticas noutros capítulos deste livro. No entanto, a apresentação das regras de sintaxe nesses capítulos sugere um modo de apresentação apropriado às construções correspondentes em programa.

3

VOCABULÁRIO: TERMOS FIXOS

Sintaxe

```

Termo = Termo-especial |
        identificador |
        ordem |
        declaração-etiqueta |
        constante inteira sem sinal |
        constante real sem sinal |
        constante-caracter |
        constante-cadeia |
Termo-especial = palavra-chave |

```

```

+ | - | * | / |
= | < > | < | <= | >= | > |
( | ) | [ | ] |
:= | . | , | ; | : | .. | ↑

```

```

Palavra-chave = and | array | begin | case | const | div |
do | downto | else | end | file | for |
function | goto | if | in | label | mod |
nil | not | of | or | packed | procedure |
program | record | repeat | set | then |
to | type | until | var | while | with

```

Identificador = letra {*(letra | algarismo)}
 Ordem = letra {*(letra | algarismo)}
 Declaração-etiqueta = {algarismo*}
 Constante inteira sem sinal = {algarismo*}
 Constante real sem sinal = {algarismo*} . {algarismo*} |
 {algarismo*} [. {algarismo*}] e [+ | -] {algarismo*}
 Constante inteira = [+ | -] Constante inteira sem sinal
 Constante real = [+ | -] Constante real sem sinal
 Constante caracter = 'caracter - imagem'
 Constante-cadeia = 'caracter-imagem {caracter-imagem*}'
 Caracter-imagem = '' | qualquer caracter excepto apóstrofo
 Letra = a | b | c | d | e | f | g | h | i | j | k | l |
 m | n | o | p | q | r | s | t | u | v | w | x | y | z |
 Algarismo = 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9
 Os termos fixos especiais são termos com significados inva-
 riáveis.

Os identificadores são usados para programas, constantes, tipos, campos de registos, variáveis, procedimentos, funções e parâmetros formais. As ordens são usadas em declarações de procedimento e de função. Nenhum identificador ou ordem pode ser escrito do mesmo modo que uma palavra-chave.

As declarações-etiquetas podem ser usadas como prefixos de declarações. As declarações-etiquetas distinguem-se pelos seus valores decimais (isto é, os zeros à esquerda não são significativos), que devem variar entre 0 e 9999.

Uma constante inteira, real ou de caracter indica um valor do tipo indicado. Usa-se a notação decimal em constantes inteiras e reais, e a letra *e* representa “vezes 10 à potência de”. Uma constante de cadeia contendo *n* caracteres-imagem (*n* > 1) indica um valor da cadeia tipo

packed array [1..n] of char

O caracter-imagem “ indica o caracter apóstrofo “”; qualquer outro caracter-imagem indica-se a si mesmo. Um final de linha pode não ser usado como caracter-imagem. Os espaços e parên-

tesis de comentário no interior de uma constante de caracteres ou cadeia são apenas caracteres-imagem.

Representações equivalentes

Excepto quando ocorrem como caracteres-imagem, os caracteres que se seguem possuem as seguintes representações equivalentes:

Caracter	Equivalente
a b c ... x y z	A B C ... X Y Z
↑	^ ou @
[	(
]	)
{	{
}	}

Exemplos

Identificadores: A número x1 v2000A CaudalOxigénio

Declarações-etiquetas: 1 999

Constantes inteiras sem sinal: 0 1 123 32767

Constantes reais sem sinal: 0.0 1.234 1e-8 0.5E+6
3.33e10

Constantes-caracteres: ' ' '*' 'A' ''

Constantes-cadeia: '*' 'Janeiro' 'Lei de Murphy'

TIPOS E OPERAÇÕES

Utilização

O “tipo” é uma propriedade fixa de cada valor, variável e função. Um tipo é caracterizado por um conjunto de valores e pelas operações que podem ser executadas sobre eles. Um valor de um tipo *não estruturado* (ordinal, real ou indicador [pointer]) é um valor único e indivisível. Um valor de um tipo *estruturado* (cadeia, registo, conjunto ou ficheiro) é composto por vários valores componentes, que podem por sua vez ser ou não estruturados.

Sintaxe

```
Tipo = Tipo-indicador | Tipo-descrição
Tipo-descrição = ordinal-descrição |
                indicador-descrição |
                cadeia-descrição [packed] |
                registo-descrição [packed] |
                conjunto-descrição [packed] |
                ficheiro-descrição [packed]
```

Os tipos normalizados são indicados por identificadores de tipo previamente definidos. Cada descrição de tipo que ocorra

num programa indica um tipo novo e diferente (mesmo que seja textualmente idêntica a outra descrição de tipo).

Existem algumas restrições aos tipos dos componentes de uma cadeia, registo ou ficheiro. No entanto, o âmbito das definições impede o uso de tipos repetidos excepto através de indicadores (pointers).

Semântica

O prefixo *packed* não tem qualquer efeito sobre a semântica de um tipo estruturado. Exige uma representação compacta para valores de tipo estruturado, mesmo à custa de um acesso menos eficaz aos componentes da estrutura. As mesmas operações são válidas para qualquer tipo estruturado *não packed* e para o correspondente tipo estruturado *packed* com as seguintes excepções: a) as operações sobre cadeias (tabela 11) são válidas apenas para certas cadeias de caracteres *packed*; b) um componente imediato de uma estrutura *packed* não pode ser usado como parâmetro de variável.

Operações

Os vários tipos usados em Pascal formam uma hierarquia (ver a figura 1). As operações válidas para qualquer classe de tipos são-no igualmente para todas as subclasses da hierarquia.

Nas tabelas 1 a 15, *A*, *B*, *C*, etc., indicam variáveis dos tipos especificados, e *a*, *b*, *c*, etc. indicam expressões secundárias dos tipos especificados.

(1) As variáveis de *todos* os tipos podem ser dadas como parâmetros de variáveis.

(2) As operações indicadas na tabela 1 são válidas para todos os tipos *T* excepto tipos de ficheiro (e tipos estruturados com componentes de ficheiro).

Por outro lado, podem-se usar valores de tudo excepto tipos ficheiro como parâmetros de valor.

(3) As operações indicadas na tabela 2 são válidas para todos os tipos não estruturados.

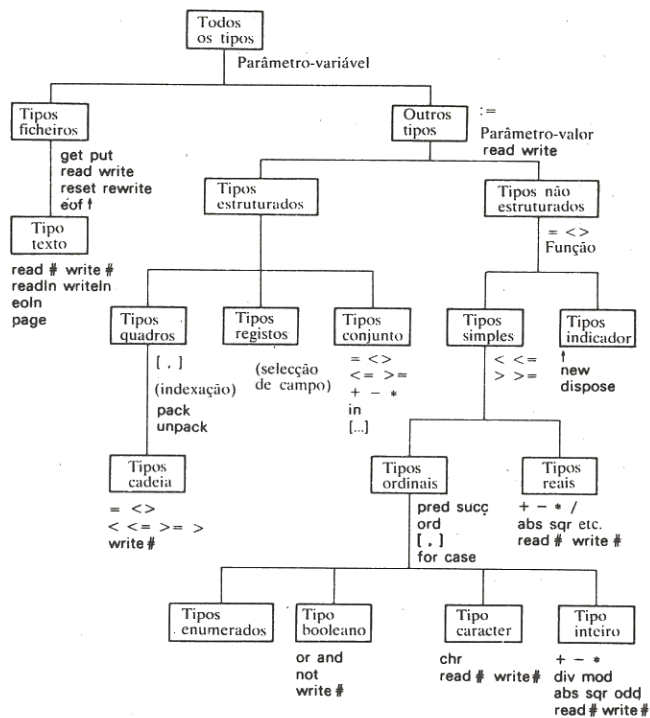


Figura 1. – Hierarquia das classes de tipos em Pascal, mostrando as operações definidas para cada classe.

indica as formas de *read* e *write* aplicáveis aos ficheiros de texto, envolvendo possivelmente uma mudança de representação.

A classe de tipos ordinais, e cada uma das suas subclasses, inclui âmbitos específicos destes tipos. As classes de tipos de quadros, de registos, de conjuntos e de ficheiros incluem os correspondentes tipos compactos (*packed*). [Agradeço a Bill Findlay, que concebeu este diagrama].

Por outro lado, podem ser apresentados como resultados de funções valores de qualquer tipo não estruturado (as operações de igualdade são também válidas para todos os tipos de cadeias e conjuntos).

Tabela 1. – OPERAÇÕES DE CÓPIA

Operação	Operando(s)	Resultado	Significado
$V := e$	$V: T$ $e: T \dagger$	–	Atribui valor de e a V (falha se o valor de e excede gama válida)
$read(F, V)$	$F: \text{file of } T$ $V: T \dagger$	–	lê componente seguinte de F para V (falha se $eof(F)$)
$write(F, e)$	$F: \text{file of } T$ $e: T \dagger$	–	Aceita para F um componente com o valor e

† Aplicam-se as regras de compatibilidade.

(4) As operações indicadas na tabela 3 são válidas para todos os tipos simples (isto é, ordinal ou *real*, mas não indicador [pointer]).

Tabela 2. – OPERAÇÕES DE IGUALDADE

Operação	Operando(s)	Resultado	Significado
$a = b$	Compatível; ou real & inteiro	Booleano	a igual a b
$a < > b$	Idem	Booleano	a diferente de b

Tabela 3. – OPERAÇÕES DE ORDENAÇÃO

Operação	Operando(s)	Resultado	Significado
$a < b$	Compatível; ou real & inteiro	Booleano	a menor do que b
$a \leq b$	Idem	Booleano	a não maior do que b
$a \geq b$	Idem	Booleano	a não menor do que b
$a > b$	Idem	Booleano	a maior do que b

Os tipos T_1 e T_2 são compatíveis se:

- (1) T_1 e T_2 pertencerem ao mesmo tipo não estruturado; ou
- (2) T_1 e T_2 formarem âmbitos de um mesmo tipo, ou um deles pertencer ao âmbito do outro; ou
- (3) T_1 e T_2 constituírem ambos tipos de cadeias com o mesmo comprimento; ou
- (4) T_1 e T_2 forem ambos tipos de conjuntos com membros compatíveis, e ambos simultaneamente *packed* ou não-*packed*.

Uma expressão do tipo T_2 é compatível com um tipo T_1 em termos de *atribuição* se:

- (1) T_1 é do mesmo tipo que T_2 , e nenhum deles é tipo-ficheiro (nem tipo estruturado com uma componente ficheiro); ou
- (2) T_1 é real e T_2 é inteiro; ou
- (3) T_1 e T_2 são tipos ordinais compatíveis; ou
- (4) T_1 e T_2 são tipos de cadeias compatíveis; ou
- (5) T_1 e T_2 são tipos de conjuntos compatíveis.

A compatibilidade de atribuição deve ser satisfeita sempre que se pretende atribuir um valor a uma variável. No caso 3, se T_1 é um tipo secundário, ou no caso 5, T_1 for um membro num tipo secundário, é possível que o valor a atribuir não seja um valor do tipo esperado, T_1 ; neste caso, o programa falha (“valor fora da gama aceite”).

5

TIPOS ORDINAIS

Utilização

Um tipo ordinal é um tipo não estruturado cujos valores são ordenados, discretos e contíguos.

Sintaxe

Ordinal-descrição = enumeração-descrição |
subdomínio-descrição

Enumeração-descrição = ({ identificador*, })

Subdomínio-descrição = constante..constante

A enumeração-descrição ($I_0, \dots, I_n$) define $I_0, \dots$ e I_n como identificadores-constantes no interior do bloco considerado.

No subdomínio-descrição $C_1..C_2$, C_1 e C_2 devem ser constantes do mesmo tipo ordinal, com $C_1 \leq C_2$.

Booleano, caracter e inteiro são tipos ordinais “standard”.

Semântica

Cada tipo ordinal define um conjunto de valores contíguos finito e completamente ordenado. Cada valor de um dado tipo ordinal possui um único número ordinal. A ordenação dos valores

é equivalente à ordenação dos seus números ordinais.

A enumeração-descrição ($I_0, \dots, I_n$) introduz um novo tipo ordinal com $n+1$ valores denotados pelos identificadores-constantes $I_0, \dots, I_n$. A ordenação destes valores é equivalente à ordem em que os identificadores são apresentados. O número ordinal de I_i é i .

Os valores do subdomínio $C_1..C_2$, onde C_1 e C_2 são constantes do tipo ordinal T, são os valores de T que se encontram entre C_1 e C_2 inclusive. $C_1..C_2$ é designado por subdomínio de T. Todas as operações válidas para operandos do tipo T são-no igualmente para operandos do tipo $C_1..C_2$, e têm o mesmo efeito exceptuando a impossibilidade de atribuir a qualquer variável ou função do tipo $C_1..C_2$ um valor fora do domínio C_1 a C_2 .

Exemplos

- (a) (Seg, Ter, Qua, Qui, Sex, Sab, Dom)
- (b) (sim, não, ignorado)
- (c) Seg..Sex
- (d) 'A'..'Z'
- (e) -intmax..intmax
- (f) (Jan, Fev, Mar, Abr, Mai, Jun, Jul, Ago, Set, Out, Nov, Dez)

Operações

Além das operações válidas para todos os tipos simples (tabelas 1 a 3), as citadas na tabela 4 são válidas para todos os tipos ordinais I. Por outro lado, os valores de qualquer tipo ordinal podem ser usados para indicição de quadros, contagem em declarações FOR ou como membros de conjuntos.

Tabela 4. – OPERAÇÕES ORDINAIS

Operação	Operando(s)	Resultado	Significado
<i>pred(a)</i>	T(ordinal)	T	Predecessor de a , isto é, um valor cujo ordinal é $ord(a)-1$ (falha se a é o valor mínimo de T)
<i>succ(a)</i>	T(ordinal)	T	Sucessor de a , isto é, um valor cujo ordinal é $ord(a)+1$ (falha se a é o valor máximo de T)
<i>ord(a)</i>	T (ordinal)	Inteiro	Número ordinal de a

Exemplos

- (a) $mês := pred(mês)$
- (b) *if* $mês \geq Jul$ *then*...
- (c) *write* (saída, $ord(mês)+1$)

O TIPO BOOLEANO

Utilização

Para introduzir as operações comuns da álgebra booleana, e para permitir a selecção condicional e a repetição de declarações.

Semântica

Booleano é um tipo ordinal “standard”. Os seus valores (e a sua ordenação) são definidos por:

Booleano = (falso, verdadeiro)

Assim, $ord(falso)=0$ e $ord(verdadeiro)=1$, de onde falso = $pred(verdadeiro)$.

Operações

Além das operações válidas para todos os tipos ordinais (tabelas 1 a 4), as operações indicadas na tabela 5 são também válidas no tipo Booleano.

Tabela 5. – OPERAÇÕES BOOLEANAS

Operação	Operando(s)	Resultado	Significado
$b \text{ or } c$	Booleano	Booleano	Soma lógica de b e c
$b \text{ and } c$	Booleano	Booleano	Produto lógico de b e c
$Not\ b$	Booleano	Booleano	Negação lógica de b
$Write\ (F,b)$	F: texto b: Booleano	—	Atribui a F “Verdadeiro” ou “Falso”

Exemplos

- (a) descoberto:=AntigoFicheiroContas ↑ .balanço < 0
- (b) IF (dia=25) AND (mês=Dez) THEN...
- (c) IF NOT (descoberto <= (produto=crédito)) THEN...
Aqui, { ' <=' significa “implica”. }

O TIPO CARACTER

Utilização

Para permitir o processamento de dados alfanuméricos.

Semântica

O tipo caracter (*Char*) é um tipo ordinal “standard”. Os seus valores são um conjunto de caracteres definidos para a aplicação considerada. Cada caracter possui um único número ordinal (“código interno”) que é definido pela aplicação. A ordenação dos caracteres é equivalente à ordenação dos seus números ordinais. Em geral, são apenas válidos os seguintes pressupostos quanto à ordenação de caracteres: (a), os algarismos “0” a “9” estão ordenados numericamente e são contíguos; e (b), as letras no interior de cada conjunto individual de letras estão ordenadas alfabeticamente mas não são necessariamente contíguas.

Operações

Além das operações já indicadas como válidas para todos os tipos ordinais (tabelas 1 a 4), são ainda válidas neste tipo as operações indicadas na tabela 6.

Tabela 6. – OPERAÇÕES DE CARACTERES

Operação	Operando(s)	Resultado	Significado
<i>chr(i)</i>	inteiro	caracter	caracter cujo número ordinal é <i>i</i> (falha se este caracter não existir)
<i>read (F,C)</i>	F: texto C: caracter	—	lê um caracter de F e atribuiu-o a C (falha se <i>eof(F)</i>)
<i>write (F,c)</i>	F:texto c:caracter	—	atribui a F o valor de c

Exemplos

- (a) inicial:=nome[1]
- (b) caracter:=*chr(ord('0')+modo contagem 8)*
- (c) IF (caracter='—') OR (caracter >='A') AND (caracter <='Z') THEN ...
- (d) READ (entrada,caracter)
- (e) WRITE (saída, caracter)

Exemplos

- (a) $\text{total} := \text{total} \div 8$
- (b) IF $i+j=N+1$ THEN...
- (c) IF impar (soma) THEN...
- (d) READ (entrada,soma)
- (e) WRITE (saída, soma+somaneg)

8

O TIPO INTEIRO

Utilização

Permitir uma aritmética exacta em números inteiros.

Semântica

O tipo Inteiro é um tipo ordinal “standard”. Os seus valores são um subconjunto finito dos números inteiros definido pela aplicação. O número ordinal de cada valor inteiro é ele próprio.

Operações

Além das operações válidas para todos os tipos ordinais (tabelas 1 a 4), as operações indicadas na tabela 7 são também válidas para o tipo Inteiro.

As operações sobre este tipo são realizadas correctamente se os seus operandos e os seus resultados se mantiverem dentro da gama entre $-intmax$ e $+intmax$ inclusive, onde $intmax$ é uma constante definida pela aplicação. Se qualquer operação der um resultado inteiro fora dos limites aceites para a aplicação, o programa falha.

Tabela 7. – OPERAÇÕES DE INTEIROS

Operação	Operando(s)	Resultado	Significado
$+i$	Inteiro	Inteiro	i
$-i$	Inteiro	Inteiro	Negação de i
$i+j$	Inteiro	Inteiro	Soma de i e j
$i-j$	Inteiro	Inteiro	Diferença de i e j
$i*j$	Inteiro	Inteiro	Produto de i e j
$i \div j$	Inteiro	Inteiro	Quociente de i por j
$i \bmod j$	Inteiro	Inteiro	i módulo j ($0 \leq i \bmod j < j$) (falha se $j \leq 0$)
$abs(i)$	Inteiro	Inteiro	Valor absoluto de i
$sqr(i)$	Inteiro	Inteiro	Quadrado de i
$odd(i)$	Inteiro	Booleano	Verdadeiro se i é ímpar
$read(F,I)$	F:texto I: Inteiro †	—	Lê uma constante inteira de F e atribui o seu valor a I (falha se não encontra uma)
$write(F,i)$	F:texto i: Inteiro	—	Atribui a F o valor de i como constante-inteira

† Aplicam-se as regras de compatibilidade de atribuição.

Exemplos

- (a) $\text{circunferência} = 2 * \pi * \text{raio}$
- (b) IF ABS((y-ycopia)/ycopia) < 1e-8 THEN ...
- (c) READ (entrada,x)
- (d) WRITE (saída,arredondamento(sqrt(sqr(x)+sqr(y))))

9

O TIPO REAL

Utilização

Permitir uma aritmética aproximada em números reais.

Semântica

O tipo Real é um tipo simples “standard”. Os seus valores são um subconjunto limitado dos números reais definido pela aplicação.

Operações

Além das operações válidas para todos os tipos simples (tabelas 1 a 3), as operações indicadas na tabela 8 são também válidas para o tipo Real.

Qualquer (sub)expressão real pode ser substituída por uma (sub)expressão inteira, exceptuando o facto de +, -, *, abs, sqr, e write com apenas operando(s) inteiros constituírem operações inteiras.

As operações reais produzem resultados aproximados, cujo erro é definido pela aplicação.

Se qualquer operação produzir um resultado real fora dos limites definidos pela aplicação, o programa falha.

Tabela 8. – OPERAÇÕES REAIS

Operação	Operando(s)	Resultado	Significado
$+r$	Real	Real	r
$-r$	Real	Real	Negação de r
$r+s$	Real	Real	Soma de r e s
$r-s$	Real	Real	Diferença de r e s
$r*s$	Real	Real	Produto de r e s
r/s	Real	Real	Quociente de r por s
$\text{abs}(r)$	Real	Real	Valor absoluto de r
$\text{sqr}(r)$	Real	Real	Quadrado de r
$\text{trunc}(r)$	Real	Inteiro	r truncado para número inteiro
$\text{round}(r)$	Real	Inteiro	r arredondado para número inteiro mais próximo
$\text{sqr}(r)$	Real	Real	Raiz quadrada de r (falha se $r < 0$)
$\text{exp}(r)$	Real	Real	e à potência r
$\text{ln}(r)$	Real	Real	Logaritmo natural de r (falha se $r \leq 0$)
$\text{sin}(r)$	Real	Real	Seno de r (radianos)
$\text{cos}(r)$	Real	Real	Cosseno de r (radianos)
$\text{arctan}(r)$	Real	Real	Inverso da tangente de r (radianos)
$\text{read}(F,R)$	F: texto R: real †	—	Lê uma constante inteira ou real de F e atribui o seu valor a R
$\text{Write}(F,r)$	F: texto r: real	—	Atribui a F o valor de r como constante real

† Aplicam-se as regras de compatibilidade de atribuição.

TIPOS DE INDICADORES

Utilização

Construção de estruturas de dados interligadas de variação dinâmica.

Sintaxe

Indicador-descrição = $\uparrow$ domínio-tipo
domínio-tipo = tipo-identificador

Semântica

Cada valor do tipo indicador (pointer) $\uparrow T$ é uma referência a uma variável anônima, cujo tipo é T , ou um valor nulo (indicado por *nil*) que não se refere a qualquer variável.

Exemplos

- (a) $\uparrow$ ListaNomes
- (b) $\uparrow$ MatrizesOrdemN

Operações

Além das operações válidas para todos os tipos não estruturados (tabelas 1 e 2), as operações indicadas na tabela 9 são também válidas para todos os tipos de indicadores $\uparrow T$.

Tabela 9. – OPERAÇÕES COM INDICADORES

Operação	Operando(s)	Resultado	Significado
$P \uparrow$	$P: \uparrow T$	Variável tipo T	A variável anônima a que P se refere (falha se $P = \text{nil}$)
$\text{new}(P)$	$P: \uparrow T$	—	Cria uma variável anônima, de tipo T e valor indefinido, e refere P a ela (ver adiante)
$\text{dispose}(P)$	$P: \uparrow T$	—	Destrói a variável a que P se refere (falha se $P = \text{nil}$) (ver adiante)

A variável anônima existe desde que é criada (através de *new*) até ser destruída (por *dispose*), ou até terminar o programa.

$\text{New}(P, c_1, \dots, c_n)$ — onde P é uma variável de tipo $\uparrow T$ e T é um tipo registrado com variantes internas correspondentes às variantes-etiquetas $c_1, \dots, c_n$ — cria uma variável anônima de tipo T , como na tabela 9. No entanto, esta variável anônima é subsequentemente restringida às variantes especificadas por estas variantes-etiquetas, e não pode ser manipulada como um conjunto; pode apenas ser usada como um registo-variável num campo ou numa declaração *WITH*. Os campos limites das variáveis não são inicializados por *new*, podendo aceitar quaisquer valores que não provoquem uma mudança de variante (estas restrições permitem minimizar a quantidade de memória atribuída à variável anônima).

$\text{Dispose}(P, l_1, \dots, l_m)$ deve ser usada para destruir uma variável

anónima criada por $new(P, c_1, \dots, c_n)$, onde $m \geq n$ e as variantes-etiquetas $l_1, \dots, l_n$ indicam as mesmas variantes internas referenciadas por $c_1, \dots, c_n$.

Exemplos

- (a) $new(ref)$
- (b) $ref \uparrow .nome := 'E. W. Dijkstra$
- (c) $ref := ref \uparrow .seguinte$
- (d) $dispose(ref)$
- (e) IF $ref < > nil$ then

11

TIPOS DE QUADROS

Utilização

Um quadro é uma estrutura de dados composta por um número fixo de componentes, todos do mesmo tipo. Qualquer componente do quadro é seleccionado por um *índice*, que pode ser processado.

Sintaxe

quadro-descrição = $array [\{ \text{índice-tipo}^*, \}]$
of tipo-componente

tipo-índice = tipo

tipo-componente = tipo

Cada tipo-índice deve ser ordinal. O tipo-componente não está sujeito a restrições.

Abreviaturas

As descrições-tipo

$array [S_1, \dots, S_n]$ of T

$packed array [S_1, \dots, S_n]$ of T

são, respectivamente, abreviaturas de:
array [S₁] *of*... *array* [S_n] *of* T
packed array [S₁] *of*... *packed array* [S_n] *of* T

Semântica

Cada valor do tipo quadro *array* [S] *of* T consiste num componente, de tipo T, para cada valor (índice) do tipo ordinal S.

Exemplos

- (a) *array* [1..N,1..N] *of real*
 { ...equivalente a *array* [1..N] *of array* [1..N] *of real* }
- (b) *array* [1..profundpágina] *of packed array*
 [1..largpágina] *of char*
- (c) *array* [0..108] *of* ↑ ListaNomes
- (d) *array* [meses] *of record*
 HorasSol,Chuva:*real*;
 NrD as :28..31
 end
- (e) *array* [char] *of* 0..intmax
- (f) *packed array* [1..100000] *of Boolean*

Operações

Além das operações indicadas na tabela 1, as citadas na tabela 10 são válidas para todos os tipos de quadros.

Tabela 10. – OPERAÇÕES DE QUADROS

Operação	Operando(s)	Resultado	Significado
A[i]	A: <i>array</i> [S] <i>of</i> T i:S†	Variável de tipo T	O componente de A cujo índice é i (A pode ser <i>packed</i>) (falha se i fora da gama)
pack(A,i,B)	A: <i>array</i> [S ₁] <i>of</i> T B: <i>packed array</i> [S ₂] <i>of</i> T i:S ₁ †	—	Copia A[i], A[succ(i)], etc. em todos os componentes de B
unpack(B,A,i)	Idem	—	Copia todos os componentes de B em A[i], A[succ(i)], etc.

† Aplicam-se as regras de compatibilidade de atribuição

Por outro lado, os valores de qualquer tipo de quadro podem ser usados como parâmetros de conformação de quadros.

Exemplos

- (a) A:=B
- (b) A[i]:=B[i+1]
- (c) S[i,j]:=A[i,j]+B[i,j]
- (d) pack(buffer,1,nome)
- (e) unpack(nome, buffer,soma+1)

TIPOS DE CADEIAS

Utilização

Facilitar o processamento de cadeias de caracteres.

Sintaxe

Um tipo cadeia é qualquer tipo quadro da seguinte forma:

packed array [1]..n] of char

(onde n é uma constante positiva).

Semântica

Cada valor deste tipo cadeia é uma cadeia de caracteres de comprimento fixo n . Os valores de cadeia com o mesmo comprimento estão ordenados do ponto de vista lexicográfico.

Exemplos

(a) *packed array [1..largpágina] of char*

Operações

Além das operações válidas para todos os tipos de quadro (tabelas 1 e 10), as operações indicadas na tabela 11 são-no igualmente para todos os tipos de cadeia.

Tabela 11. – OPERAÇÕES DE CADEIAS

Operação	Operando(s)	Resultado	Significado
$s=t$	Compatível	Booleano	s igual a t
$s<>t$	Compatível	Booleano	s diferente de t
$s<t$	Compatível	Booleano	Ordenação lexicográfica, baseada na ordenação de caracteres individuais atribui a F todos os caracteres n de s
$s\leq t$	Compatível	Booleano	
$s\geq t$	Compatível	Booleano	
$s>t$	Compatível	Booleano	
$write(F,s)$	F :texto s :cadeia de comprimento n	—	

Exemplos

- (a) nome:='Donald Knuth'
- (b) nome [16]:='.'
- (c) IF nome<>Ant.FicheiroContas ↑ .possuidor THEN ...
- (d) WRITE (saída,nome)

TIPOS DE REGISTOS

Utilização

Um registo é uma estrutura de dados composta por um número fixo de componentes (*campos*), que podem ser de diferentes tipos; ou de um número fixo de campos juntamente com uma *parte variante*, cuja estrutura pode ser alterada dinamicamente. Os campos são escolhidos pelo nome.

Sintaxe

```
registo-descrição=registo
                  campo-lista
                  fim
campo-lista=
    [parte-fixa [:]] |
    [parte-fixa;]
    parte-variante[:]
parte-fixa=
    { registo-seccção*; }
registo-seccção=
    { identificador*, } :tipo
```

```
parte-variante= case secção-limite of { variante*; }
secção-limite=[identificador:] tipo-identificador
variante= { variante-etiqueta*, } : (variante-campo-lista)
variante-etiqueta=constante
variante-campo-lista=campo-lista
```

Cada identificador à esquerda de ':' numa secção-registo ou secção-limite é definido como um identificador-campo do tipo registo. Todos os identificadores-campo de um tipo registo podem ser diferentes entre si, mas não o são necessariamente de quaisquer identificadores-campo de outro tipo registo ou de quaisquer outros identificadores. Os identificadores-campo só podem ser usados em designadores-campo.

O identificador-tipo na secção-limite de uma parte-variante deve ser um ordinal de tipo T, e todas as variantes-etiquetas na parte-variante devem ser constantes do tipo T, cada valor de T ocorrendo exactamente uma vez.

Semântica

Os valores de um tipo-registo são definidos pelo seu campo-lista. Cada valor de registo consiste num campo correspondente a cada identificador-campo definido na sua parte-fixa (se existir), juntamente com o valor da sua parte-variante (se existir).

Cada valor da parte-variante

```
case I:of
    I11,...:(VFL1);
    In1,...:(VFLn)
```

consiste num *campo limite*, designado por I, juntamente com os campos do campo-lista-variante único VFL_i (Variant-Field-List), correspondente ao valor actual I_{ik} do campo-limite. Ocorre uma mudança de variante quando se atribui ao campo limite um novo valor correspondente a algum campo-lista-variante diferente VFL_j.

Cada valor da parte-variante

```
case T of
```


$l_{11}, \dots, (VFL_1);$
 $l_{n1}, \dots, (VFL_n)$

onde a secção-limite não define qualquer identificador-campo, consistindo num dos campos-listas-variantes VFL_i (não existe campo-limite). Ocorre uma mudança de variante imediatamente antes de um acesso a qualquer campo de um campo-lista-variante VFL_j diferente de VFL_i .

Quando ocorre uma mudança de variante, os campos do antigo campo-lista-variante VFL_i deixam de existir, e os campos do novo campo-lista-variante VFL_j passam a existir com valores indefinidos. Qualquer tentativa subsequente de aceder a um dos campos de VFL_i equivale a um erro.

Exemplos

- (a) *registo*
 $R, I: \text{real}$
 fim
- (b) *registo compacto*
 $\text{dia}: 1..31;$
 $\text{mês}: \text{Meses};$
 $\text{ano}: \text{inteiro}$
 fim
- (c) *registo*
 $\text{ident} : \text{Pessoais};$
 $\text{nome} : \text{CadeiasNome};$
 $\text{apelido} : [1.3] \text{CadeiasNome};$
 $\text{data de nascimento}: \text{Datas};$
 $\text{sexo} : (\text{masculino}, \text{feminino});$
 $\text{estatuto} : \text{EstatutoMatrimonial de}$
 $\text{solteiro}: ();$
 $\text{casado}: (\text{datacasamento}: \text{Datas};$
 $\text{cônjuge}: \text{Pessoais});$
 fim

Operações

Além das citadas na tabela 1, a operação indicada na tabela 12 é válida para todos os tipos de registos.

Tabela 12. – OPERAÇÃO COM REGISTOS

Operação	Operando(s)	Resultado	Significado
R.F	R:registo F:T; fim	Variável tipo T	O campo F de R (R pode ser compacto)

Exemplos

- (a) $\text{datanasc} := \text{hoje}$
- (b) $\text{NovoFicheiroContas} \uparrow := \text{AntigoFicheiroContas} \uparrow$
- (c) $\text{hoje.ano} := 1980$
- (d) $\text{max} := \text{MêsTabela}[\text{meses}].\text{HorasSol}$

TIPOS DE CONJUNTO

Utilização

Um conjunto é uma colecção não ordenada de membros diferentes, todos do mesmo tipo.

Sintaxe

Descrição-conjunto=conjunto de tipos-membro

Tipo-membro=tipo

O tipo-membro deve ser ordinal. A aplicação pode restringir a gama de valores do tipo-membro.

Semântica

Cada valor do tipo-conjunto conjunto de T é um subconjunto dos valores do tipo-membro T. Cada tipo-conjunto inclui um valor de conjunto vazio.

Exemplos

- (a) conjunto de caracteres
- (b) conjunto de(vermelho,amarelo,azul)
- (c) conjunto de 1..N
- (d) conjunto de meses

Operações

Além das citadas na tabela 1, as operações indicadas na tabela 13 são válidas para todos os tipos de conjuntos.

Tabela 13. – OPERAÇÕES DE CONJUNTOS

Operação	Operando(s)	Resultado	Significado
$s = t$	Compatível	Booleano	s igual a t
$s < > t$	Compatível	Booleano	s diferente de t
$s < = t$	Compatível	Booleano	s incluído em t
$s > = t$	Compatível	Booleano	s inclui t
$m \text{ em } s$	$m: T$ $s: \text{conjunto de } T$	Booleano	Verdadeiro se m é um membro de s (s pode ser compacto)
$s + t$	Compatível	O mesmo	União de s e t
$s - t$	Compatível	O mesmo	Diferença de s e t
$s * t$	Compatível	O mesmo	Intersecção de s e t
$[m]$	Ordinal, T	Conjunto de T	Conjunto de um elemento contendo o valor de m
$[m..n]$	Ordinal, T	Conjunto de T	Conjunto contendo todos os valores entre m e n inclusive; vazio se $m > n$

O tipo de $[m]$ ou $[m..n]$ pode ser compacto ou não, dependendo do contexto.

Se qualquer operação de conjuntos produz um conjunto com membros fora dos limites permitidos na aplicação, o programa falha.

Exemplos

- (a) meses30dias:=[Abri,Jun,Set,Nov]
- (b) mesescompridos:=[Jan..Dez] - meses30dias - [Fev]
- (c) Se mês incluído em meses30dias, então...
- (d) Se mesescompridos $\geq$ [Jul..mês] então...

TIPOS DE FICHEIROS

Utilização

Um ficheiro é uma estrutura de dados em série composta por um número variável de componentes, todos do mesmo tipo. Ao contrário das outras estruturas de dados de Pascal, um ficheiro será normalmente armazenado fora da memória primária, e os seus componentes não podem ser acedidos aleatoriamente.

Sintaxe

Descrição-ficheiro = ficheiro de tipos-componente

Tipo-componente = tipo

O tipo-componente não deve ser ele próprio de tipo-ficheiro (nem um tipo estruturado com um componente de ficheiro).

Semântica

Cada valor do tipo-ficheiro *ficheiro de T* é uma sequência de zero ou mais componentes, que são todos do tipo T. Durante o processamento, o ficheiro encontra-se em modo leitura ou modo escrita, e possui uma posição leitura/escrita actual. Em

modo escrita, a posição escrita encontra-se sempre na extremidade do ficheiro.

Em qualquer momento só um componente do ficheiro F é acessível, em particular o componente designado pela posição leitura/escrita. Este componente “actual” de F é acessível através de um “buffer” ficheiro associado a F.

Exemplos

- (a) ficheiro de inteiros
- (b) ficheiro de quadros compactos [1..80] de caracteres
- (c) ficheiro de Transacções

Operações

As operações indicadas na Tabela 14 são válidas para todos os tipos ficheiro.

Reset (F) (respectivamente *get*(F)) copia o primeiro (seguinte) componente de F para $F \uparrow$. Se este componente não existe, $F \uparrow$ fica indefinido e *eof*(F) torna-se verdadeiro.

Read(F,V) é exactamente equivalente a

begin V:= $F \uparrow$; *get* (F) *end*

Write(F,e) é exactamente equivalente a:

begin $F \uparrow$:=e; *put*(F) *end*

Abreviaturas

Read(F,V₁,...,V_n) é uma abreviatura de:

begin read(F,V₁);...; *read*(F,V_n) *end*

Write(F,e₁,...,e_n) é uma abreviatura de:

begin write(F,e₁);...; *write*(F,e_n)*end*

Tabela 14. – OPERAÇÕES COM FICHEIROS

Operação	Operando(s)	Resultado	Significado
$F \uparrow$	F:ficheiro de T	Variável de tipo T	Buffer ficheiro associado a F
$eof(F)$	F:ficheiro de T	Booleano	Verdadeiro se a posição de F está para além do último componente ($F \uparrow$ é indefinido)
$reset(F)$	F:ficheiro de T	–	Conduz a posição de leitura de F para o primeiro componente, actualiza $F \uparrow$ ou $eof(F)$ (ver texto)
$get(F)$	F:ficheiro de T	–	Avança a posição de F de um componente, actualiza F ou $eof(F) \uparrow$ (ver texto) (falha se $eof(F)$ ou se está em modo escrita)
$read(F,V)$	F:ficheiro de T V:T †	–	Lê o componente seguinte de F para V (ver texto)
$rewrite(F)$	F:ficheiro de T	–	Conduz posição de escrita de F para o primeiro componente, isto é, F é esvaziado; $F \uparrow$ torna-se indefinido.
$put(F)$	F:ficheiro de T	–	Passa conteúdo de $F \uparrow$ para F; $F \uparrow$ torna-se indefinido (falha se está em modo leitura)
$write(F,e)$	F:ficheiro de T e:T †	–	Escreve em F um componente com o valor e (ver texto).

† Aplicam-se as regras de compatibilidade de atribuição.

Exemplos

Ver o Capítulo 41, na página 105, dedicado a programas.

16

O TIPO TEXTO

Utilização

Facilitar entradas/saídas legíveis.

Semântica

Texto é um tipo ficheiro standard. Cada valor do tipo texto é um ficheiro de caracteres agrupados em linhas. Cada linha de um ficheiro de texto é vista como uma sequência de valores “caracter” seguidos de um componente de fim de ficheiro. Apenas em modo escrita, a última linha de um ficheiro de texto pode estar incompleta, na medida em que o seu fim de linha pode não ter sido ainda acrescentado.

Operações

As operações de ficheiro (tabela 14) são todas válidas para um ficheiro de texto F, e possuem efeitos semelhantes com uma excepção: quando a posição de leitura é deslocada por *reset* ou *get* (ou *read*) para um fim de linha, é colocado um caracter em branco em $F \uparrow$. As tabelas 15(a) e 15(b) mostram as operações adicionais válidas para os ficheiros de texto.

Tabela 15(a)
OPERAÇÕES DE FICHEIRO DE TEXTO (LEITURA)

Operação	Operando(s)	Resultado	Significado
<i>read</i> (F,C)	F:texto C:caracteres +	—	Lê um caracter de F e atribuiu-o a C (ver tabela 14)
<i>read</i> (F,I)	F:texto I:inteiro +	—	Lê uma constante inteira de F e atribui o seu valor a I (ver adiante)
<i>Read</i> (F,R)	F:texto R:real +	—	Lê uma constante inteira ou real de F e atribui o seu valor a R (ver abaixo)
<i>readln</i> (F)	F:texto	—	Passa o fim da linha seguinte (ver abaixo)
<i>eoln</i> (F)	F:texto	Booleano	Verdadeiro se a posição leitura de F designa um fim de linha (implica $F \uparrow = ' '$)

+ Aplicam-se as regras de compatibilidade de atribuição.

Read(F,I) ou *read*(F,R) falha se *eof*(F) se torna verdadeiro durante a operação, ou se a sequência de caracteres lidos não forma uma constante inteira ou real. Quaisquer posições em branco ou fins de linha precedentes são eliminados. Depois da operação, $F \uparrow$ contém o caracter que se segue imediatamente à constante inteira ou real.

Readln(F) é exactamente equivalente a:

begin while not eoln(F) do get(F);get(F) end

Tabela 15(b)
OPERAÇÕES DE FICHEIRO DE TEXTO (ESCRITA)

Operação	Operando(s)	Resultado	Significado
<i>write(F,e:w)</i>	F:texto e:caracter/ cadeia/ booleano/ inteiro/ real	—	Escreve a representação do valor de <i>e</i> , justificando à direita num campo de <i>w</i> espaços (ou mais se necessário) (ver adiante)
<i>write(F,e)</i>	w:inteiro F:texto e: como acima	—	(falha se $w < 1$) Como <i>write(F,e:w)</i> , com valor previsto para <i>w</i> (ver adiante)
<i>write(F,e:w:f)</i>	F:texto e:real w:inteiro f:inteiro	—	Escreve representação de vírgula fixa do valor de <i>e</i> , com <i>f</i> casas decimais, num campo de <i>w</i> espaços (mais se necessário) (falha se $w < 1$ ou $f < 1$)
<i>writeln(F)</i>	F:texto	—	Escreve um fim de linha para F
<i>page(F)</i>	F:texto	—	Inicia uma página nova quando F é impresso

Em *write(F,e...)* a representação escrita e o valor previsto de *w* dependem do tipo de *e*:

- (1) caracteres: é escrito o próprio valor do caracter. O *w* previsto é 1.
- (2) cadeia de comprimento *n*: são escritos todos os *n* caracteres da cadeia se $w \geq n$; senão são apenas escritos os primeiros *w* caracteres. O *w* previsto é *n*.
- (3) booleano: é escrita uma das cadeias 'verdadeiro' ou 'falso'. O *w* previsto é definido pela aplicação.
- (4) inteiro: é escrita uma constante inteira (com o sinal suprimido se for positivo). O *w* previsto é definido pela aplicação.
- (5) real: é escrita uma constante real, como se segue:
Write(F,e:w) ou *write(F,e)* exige uma representação em vírgula flutuante, da forma

(em branco | -)algarismo. { algarismo* } *e*(+|-) { algarismo* } onde o número de algarismos do expoente (s) é fixo e definido pela aplicação, e o número de algarismos da fracção é o menor de *w-s-5* e 1. O *w* previsto é definido pela aplicação.

Write(F,e:w:f) exige uma representação de vírgula fixa, da forma

$$[-] \{ \text{algarismo}^* \} . \{ \text{algarismo}^* \}$$

onde o número de algarismos da fracção é *f*.

Abreviaturas

Read(F,V₁,...V_n) é uma abreviatura de:

begin read(F,V₁);...;read(F,V_n)end

Readln(F,V₁,...V_n) é uma abreviatura de:

begin read(F,V₁);...;read(F,V_n);readln(F)end

Write(F,p₁,...,p_n) é uma abreviatura de:

begin write(F,p₁);...;write(F,p_n)end

Writeln(F,p₁,...,p_n) é uma abreviatura de:

begin write(F,p₁);...;write(F,p_n);writeln(F)end

onde cada *p_i* designa qualquer das formas *e:w,e*, ou *e:w:f* (ver a tabela 15(b)).

Se o parâmetro de ficheiro de *read*, *readln*, *eoln* ou *eof* é omitido, é considerado válido o ficheiro de entrada standard. Se o parâmetro de ficheiro de *write*, *writeln* ou *page* é omitido, é considerado válido o ficheiro de saída standard.

Exemplos

Ver em "Programas".

BLOCOS

Utilização

Executar uma sequência de declarações usando constantes, tipos, variáveis, procedimentos e funções declaradas localmente. Um bloco faz parte de um programa, de um procedimento ou de uma função.

Sintaxe

Bloco=

- parte de declaração de etiqueta
- parte de definição de constante
- parte de definição de tipo
- parte de declaração de variável
- parte de procedimento e função
- parte de instrução

Parte de definição de constante=

[const { definição de constante;* }]

Parte de definição de tipo=

[tipo { definição de tipo;* }]

Parte de declaração de variável=

[var { declaração de variável;* }]

Parte de procedimento e função=
 { *(declaração de procedimento |
 declaração de função); }

Parte de instrução=
 declaração composta

Semântica

Um bloco é executado efectuando a sua parte de instrução. As variáveis declaradas na parte de declarações de variável (excepto quaisquer que sejam parâmetros de programa) só existem durante a execução do bloco, e os seus valores finais são indefinidos.

Exemplos

Ver em “Procedimentos”, “Funções” e “Programas”.

Exemplos

Os identificadores de constante definidos nestes exemplos são usados noutros exemplos deste livro.

- (a) largpagina=132
- (b) profpagina=66
- (c) branco=' '
- (d) pi=3.1415926536
- (e) título='Pascal'
- (f) AbsZero=-273
- (g) intmin=-intmax
- (h) pergunta='?'
- (i) N=10
- (j) AD=verdadeiro
- (k) separador=branco
- (l) eps=0.5e-8

18

DEFINIÇÕES DE CONSTANTES

Utilização

Introduzir constantes nomeadas.

Sintaxe

Definição de constante=

identificador=constante

Constante=[+ | -] identificador de constante |
[+ | -] constante inteira sem sinal |
[+ | -] constante real sem sinal |
constante-caracter |
constante-cadeia

A definição de constante $I=c$ define I como um identificador de constante, denotando a constante c , no interior do bloco mais pequeno que a contém. I não pode ocorrer em c .

Um identificador de constante a seguir a um “+” ou “-” deve ser do tipo inteiro ou real.

Identificador de constante predefinido

$intmax$ =maior inteiro positivo para o qual as operações entre inteiros são realizados correctamente.

DEFINIÇÕES DE TIPO

Utilização

Introduzir tipos nomeados.

Sintaxe

Definição de tipo =
Identificador=tipo

A definição de tipo $I=T$ define I como um identificador de tipo, denotando o tipo T , no interior do bloco mais pequeno que o contém. I pode não ocorrer em T , excepto no caso de uma descrição de indicador (*pointer*) $\uparrow I$.

Identificadores de tipo predefinidos

Booleano = (falso, verdadeiro);
caracter = um conjunto de caracteres;
inteiro = um subconjunto finito de números inteiros;
real = um subconjunto limitado de números reais;
texto = ficheiro de caracteres agrupados em linhas.

Exemplos

Os identificadores de tipo definidos nos exemplos seguintes são usados noutros exemplos deste livro.

- (a) $\text{Pessoais} = 0..999999999$
- (b) $\text{Meses} = (\text{Jan}, \text{Fev}, \text{Mar}, \text{Abr}, \text{Mai}, \text{Jun}, \text{Jul}, \text{Ago}, \text{Set}, \text{Out}, \text{Nov}, \text{Dez})$
- (c) $\text{Datas} = \text{registo compacto}$
 $\text{dia} : 1..31;$
 $\text{mês} : \text{Meses};$
 $\text{ano} : \text{inteiro}$
 fim
- (d) $\text{CadeiasNome} = \text{quadro compacto } [1..16] \text{ de caracteres}$
- (e) $\text{Transacções} = \text{registo}$
 $\text{data} : \text{Datas};$
 $\text{conta} : 0..999999;$
 $\text{tipo} : (\text{crédito}, \text{débito});$
 $\text{quantia} : 1..intmax$
 fim
- (f) $\text{Estadomatrimonial} = (\text{solteiro}, \text{casado}, \text{divorciado}, \text{viúvo})$
- (g) $\text{Refs.ListaNomes} = \uparrow \text{NodosListaNomes}$
- (h) $\text{NodosListaNomes} = \text{registo}$
 $\text{nome} : \text{CadeiasNome};$
 $\text{seguinte} : \text{RefsListaNomes}$
 fim
- (i) $\text{MatrizesOrdemN} = \text{quadro } [1..N, 1..N] \text{ dos reais}$

DECLARAÇÕES DE VARIÁVEIS

Utilização

Introduzir variáveis nomeadas.

Sintaxe

Declaração de variável=

{ identificador*, } :tipo

A declaração de variável $I_1, \dots, I_n : T$ define $I_1, \dots$ e I_n como identificadores de variáveis, de tipo T , no interior do bloco mais pequeno que os contém.

Semântica

Uma variável distinta de tipo T é introduzida para cada elemento de $I_1, \dots$ e I_n . Cada uma destas variáveis (a menos que se trate de um parâmetro de programa) existe apenas durante a execução do bloco mais pequeno que as contém, sendo o seu valor inicial indefinido.

Exemplos

Os identificadores de variável declarados nestes exemplos são usados noutros exemplos deste livro.

- (a) soma, somaneg, potência: 0..intamax
- (b) excedido: booleano
- (c) inicial, caracter: caracter
- (d) raio, circunferência, x, y, ycópia, max, min: real
- (e) mês: Meses
- (f) lista, ref: RefsListaNomes
- (g) A, B, S: MatrizesOrdemN
- (h) TabelaMeses: quadro [Meses] de registos
 - HorasSol, Chuva: real;
 - NrDias : 28..31
 - fim
- (i) nome: CadeiasNome
- (j) C1, C2, C3: registo
 - parte real, parteimag: real
 - fim
- (k) hoje, dianasc: Datas
- (l) NomesOrdens: conjunto de 'A'..'Z'
- (m) FicheiroTransacções: ficheiro de Transacções
- (n) AntFicheiroContas,
 - NovoFicheiroContas: ficheiro de registos
 - conta: 0..999999;
 - possuidor : CadeiasNome;
 - balanço: inteiro
 - fim
- (o) anttexto, novotexto, monttexto: texto
- (p) i, j: 1..N
- (q) c: 1..16
- (r) buffer: quadro [1..256] de caracteres
- (s) meses30dias, mesescompridos: conjunto de meses
- (t) op: 0..7

instrução *repeat* |
instrução *for* |
instrução *with*

Prefixar uma instrução com uma instrução-etiqueta permite referenciar a ela uma instrução *goto*.

Semântica

Uma instrução falsa, que não contenha nomes fixos (*tokens*) não possui qualquer efeito.

21

INSTRUÇÕES

Utilização

Alterar o estado do processamento, por exemplo alterar os valores das variáveis, realizar entradas/saídas, etc. As instruções podem ser simples ou estruturadas. As instruções estruturadas são compostas por outras instruções, que podem em si mesmas ser simples ou estruturadas.

Sintaxe

Instrução=
 [instrução-etiqueta:]
 (instrução simples | instrução estruturada)
instrução simples=instrução-atribuição |
 instrução *goto* |
 instrução *procedure* |
 instrução falsa
instrução falsa=
instrução estruturada=instrução composta |
 instrução *if* |
 instrução *case* |
 instrução *while* |

DECLARAÇÕES DE ATRIBUIÇÃO

Utilização

Inicializar ou alterar o valor de uma variável, ou dar um valor a uma função.

Sintaxe

Declaração de atribuição=
 variável:=expressão |
 declaração de atribuição de função
 Declaração de atribuição de função=
 Identificador de função:=expressão

A expressão deve ser compatível em termos de atribuição com o tipo da variável ou função.

A declaração de atribuições de função $F:=e$ só é permitida no interior da parte de declarações do bloco de funções da função F .

Semântica

A expressão é avaliada e o seu valor é atribuído à variável

ou função. Se o valor da expressão se encontra fora da gama aceite, o programa falha (“valor fora da gama”). A ordem de estabelecimento da identidade da variável (por exemplo processando quaisquer expressões-índice) e de avaliação da expressão é definida pela aplicação.

Exemplos

- (a) soma:=soma+1
- (b) sobrecarregado:=NovoFicheiroContas ↑.balanço < 0
- (c) caracter:=chr(ord('0')+somamod10)
- (d) C1.partimag:=1
- (e) circunferência:=2*pi*raio
- (f) mês:=Abr
- (g) ref:=nil
- (h) ref:=ref ↑.seguinte
- (i) A:=B
- (j) A[i]:=B[j]
- (k) nome:='J.Bloggs
- (l) FicheiroTransacções ↑.data:=hoje
- (m) NomesOrdens=['E','D','I','S']

```

y:=x;
x:= cópiay
fim
(b) início
datanasc.dia :=5;
datanasc.mês:=Maio;
datanasc.ano :=1978
fim

```

23

INSTRUÇÕES COMPOSTAS

Utilização

Agrupar uma sequência de instruções numa instrução única.

Sintaxe

```

Instrução-composta=
    início
    sequência de instruções
    fim
Sequência de instruções=
    { instruções*; }

```

Semântica

As instruções de uma sequência de instruções são executadas por uma ordem textual.

Exemplos

```

(a) início
    cópiay:=y;

```

INSTRUÇÕES IF

Utilização

Escolher uma instrução entre duas alternativas, baseando-se num valor booleano obtido no processamento.

Sintaxe

Instrução *if*=

IF (se) condição THEN (então)
declaração
[ELSE (senão) declaração]
Condição=expressão

Uma condição é uma expressão do tipo booleano.

Qualquer ambiguidade envolvendo instruções IF é resolvida referenciando cada ELSE à IF mais próxima. Por exemplo:

IF c_1 THEN IF c_2 THEN S_1 ELSE S_2
é interpretado como

IF c_1 THEN
início IF c_2 THEN S_1 ELSE S_2 fim

Semântica

A declaração *if* IF c THEN S_1 ELSE S_2 é executada do seguinte modo:

- (1) É avaliada a condição c ;
- (2) Se o valor de c é verdadeiro, é executado S_1 ;
- (3) Se o valor de c é falso, é executado S_2 .

A declaração *if* IF c THEN S_1 é executada do mesmo modo, sendo apenas eliminado o terceiro passo.

Exemplos

- (a) IF $x > y$ THEN
início max:=x; min:=y fim
ELSE
início max:=y; min:=x fim
- (b) IF $i > j$ THEN
IF $A[i,j] \geq 0$ THEN { A identificação reflecte }
soma:=soma+1 { aqui a correcta }
ELSE { interpretação desta }
somaneg:=somaneg+1 { instrução *if* }

DECLARAÇÕES DE CASO

Utilização

Seleccionar uma instrução entre várias alternativas, partindo de um valor ordinal obtido no processamento.

Sintaxe

Declaração de caso=

```
case (caso)índice-caso of (de)
    { membro-caso*; } [;]
```

fim

índice-caso=expressão

membro-caso=

```
{ etiqueta-caso*, } :declaração
```

etiqueta-caso = constante

O tipo índice-caso deve ser ordinal, e todos os casos-etiqueta na declaração de caso devem ser constantes diferentes deste tipo.

Semântica

A declaração de caso

```
case c of
```

```
    l11,...:S1;
```

```
    .....
```

l_{n1},...:S_n

fim

é executada do seguinte modo:

- (1) é avaliado o índice-caso *c*;
- (2) se uma das etiquetas-caso *l_{ik}* tem o mesmo valor de *c*, então só é executada a correspondente declaração *S_i*;
- (3) se não se encontrar presente qualquer etiqueta-caso, o programa falha.

Exemplos

- (a) *case* hoje.mês *of*
 Jan,Mar,Mai,Jul,Ago,Out,Dez:
 NrDias:=31;
 Abr,Jun,Set,Nov:
 NrDias:=30;
 Fev:
 if anobissextto(hoje.ano)*then*
 NrDias:=29
 else
 NrDias:=28
 fim
- (b) *if* inicial em NomesOrdens *then*
 case inicial *of*
 'T':ObedecerInserção;
 'D':ObedecerEliminação;
 'S':ObedecerSubstituição;
 'E':Terminar
 fim
- (c) *case* opcional *of*
 0: x:=y;
 1: x:=x+y;
 2: x:=x-y;
 3: x:=x*y;
 4: x:=x/y;
 5, 6, 7: { simulado }
 fim

- (b) *while* nome [c] <> em branco *do*
 início *write*(nome[c]);c:=*succ*(c) fim
- (c) *while* ref <> nula *do*
 início
 writeln(ref ↑ .nome);
 ref:=ref ↑ .seguinte
 fim

26

INSTRUÇÕES WHILE

Utilização

Executar uma declaração repetidamente, quando são possíveis zero ou mais repetições.

Sintaxe

Instrução *while*=
 while (enquanto) condição *do* (fazer) declaração

Semântica

A instrução *while c do S* é executada do seguinte modo:

- (1) é avaliada a condição *c*;
- (2) se *c* é verdadeira, *S* é executada e em seguida é-o novamente toda a declaração *while*;
- (3) se *c* é falsa, é terminada a execução da declaração *while*.

Exemplos

- (a) *while* potência < N *do*
 potência:=10*potência

INSTRUÇÕES REPEAT

Utilização

Executar uma declaração (sequência) repetidamente, onde é necessária pelo menos uma repetição.

Sintaxe

Declaração *repeat*=
 repeat (repetir) sequência de instruções
 until (até) condição

Semântica

A declaração *repeat S until c* é executada do seguinte modo:

- (1) é executada a sequência de instruções;
- (2) é avaliada a condição *c*;
- (3) se o valor de *c* é falso, toda a declaração *repeat* é novamente executada;
- (4) se o valor de *c* é verdadeiro, é terminada a execução da declaração *repeat*.

Exemplos

(a) *repeat*

cópiay:=*y*;

y:=(*y*+*x*/*y*)/2

until abs ((*y*-*cópiay*)/*cópiay*) <=1e-8

(b) *repeat*

read(*character*)

until *character* < > em branco

(c) *repeat*

c:=*succ*(*c*);

read(*nome*[*c*])

until(*c*=16) *or* *eoln or* (*nome*[*c*]= em branco)

INSTRUÇÕES FOR

Utilização

Para executar uma declaração repetidamente quando o número de repetições é conhecido de avanço.

Sintaxe

Declaração *for*=

for (para) variável de comando:=expressão inicial
(*to* | *downto*) (até) expressão final *do* (fazer)

declaração

Variável de comando=variável-identificador

expressão inicial=expressão

expressão final=expressão

A variável de comando deve ser declarada na parte de declaração de variáveis do bloco mais pequeno que a contém. O seu tipo deve ser ordinal, e as expressões inicial e final devem ser compatíveis em termos de atribuição com este tipo.

Semântica

A variável de comando é usada para contar as repetições.

A declaração *for V:=i to f do S* é executada do seguinte modo:

- (1) são avaliadas *i* e *f*;
- (2) se $i > f$, é terminada a execução da declaração *for*;
- (3) *S* é executada ($\text{ord}(f) - \text{ord}(i) + 1$) vezes, com *V* assumindo *i* valores consecutivos, $\text{succ}(i), \dots, f$ em repetições sucessivas.

A declaração *for V:=downto f do S* é executada do seguinte modo:

- (1) são avaliadas *i* e *f*
- (2) se $i < f$, é terminada a execução da declaração *for*;
- (3) *S* é executada ($\text{ord}(f) - \text{ord}(i) + 1$) vezes, com *V* assumindo *i* valores consecutivos, $i, \text{pred}(i), \dots, f$ em repetições sucessivas.

A ordem de avaliação de *i* e de *f* depende da aplicação. Se é atingido o passo 3, e *i* ou *f* estão fora da gama considerada, o programa falha. Durante a execução do passo 3, qualquer atribuição a *V* constitui um erro. Depois da terminação normal da declaração *for*, o valor de *V* é indefinido.

Exemplos

- (a) *for* mês:=Jan *to* Dez *do*
 with Tabela Meses mês *do writeln* (HorasSol,Chuva)
- (b) *for* i:=1 *to* N *do*
 for j:=1 *to* N *do*
 A[i,j]:=0
- (c) *for* c:=16 *downto* soma *do*
 nome[c]:=nome[pred(c)]

Abreviaturas

A declaração *with*
with V₁,...,V_n do S
é uma abreviatura de:
with V₁ do
.....
with V_n do S

Exemplos

- (a) *with datanasc do*
início dia:=5; mês:=Maio; ano:=1978 fim
- (b) *with NovoFicheiroContas ↑ , Ficheiro Transacções ↑ do*
case categoria of
crédito: balanço:=balanço+quantidade;
débito: balanço:=balanço-quantidade
fim
- (c) *with TabelaMeses [mês] do*
início HorasSol:=0; Chuva:=0 fim

29

INSTRUÇÕES WITH

Utilização

Permitir referenciar campos de variáveis-registos sem qualificação.

Sintaxe

Declaração *with*=
with (com) { variável-registo*, } *do* (fazer)
bloco-*with*
bloco-*with*=declaração

Na declaração *with V do S*, qualquer identificador de campo F do tipo registo de V pode actuar no interior de S como designador de campo, referenciando então V.F.

Semântica

A identidade de V é estabelecida (isto é, são processadas quaisquer expressões-índice) antes de S ser executada.

INSTRUÇÕES E ETIQUETAS GOTO

Utilização

Permitir saltos irregulares da execução. O uso disciplinado das instruções goto deve confinar-se à saída de ciclos, procedimentos e funções.

Sintaxe

Declarações goto=*goto* (ir para) instrução-etiqueta
 parte de declaração de etiqueta=
 [etiqueta { instrução-etiqueta*, }]

Todas as instruções-etiquetas que prefixam instruções na parte de declarações de um bloco devem ser diferentes e ocorrer na parte de declaração de etiquetas desse bloco; e vice-versa.

As instruções-etiquetas obedecem a regras de âmbito semelhantes às usadas para os identificadores.

- (1) Na instrução L:S *goto* L pode encontrar-se em qualquer ponto no interior de S.
- (2) Na sequência de instruções $S_1; \dots; S_n$, *goto* L pode encontrar-se em qualquer ponto no interior da sequência de instruções.

- (3) Se início $S_1; \dots; L:S_i; \dots; S_n$ fim é a parte de declarações de um bloco B, então *goto* L pode encontrar-se em qualquer ponto no interior de B (incluindo quaisquer blocos onde seja declarada uma etiqueta semelhante).

Semântica

O efeito de *goto* L consiste em continuar a execução pela instrução prefixada por L. Se L é declarado no exterior do bloco que contém *goto* L, então todas as variáveis declaradas nesses blocos (e em quaisquer outros blocos intervenientes) deixam de existir.

Não é possível que uma instrução *goto* provoque um salto para o interior de um bloco ou de qualquer declaração estruturada, nem de um membro ou de uma declaração *if* ou *case* para outro.

Exemplos

- (a) função triangular (Matriz:MatrizesOrdemN):Booleano;
 etiqueta 9;
 var
 linha,coluna:1..N ;
 início
 for linha:=2 to N do
 for coluna:=1 to linha-1 do
 if Matriz [linha,coluna]<>0 then
 início triangular:=falso;*goto* 9 fim;
 triangular:=verdadeiro;
 9:fim { triangular }
- (b) programa esqueleto;
 etiqueta 999;

 procedimento nome;


```

início
.....
goto 999  { parar }
.....
fim  { nome } ;
início  esqueleto
.....
nome;
.....
999:
fim.

```

31

EXPRESSÕES

Utilização

Uma expressão é uma regra para processar um valor.

Sintaxe

```

expressão=expressão simples |
           expressão simples (=|<|>|<|<=|>=|>|in|
           expressão simples
expressão simples=[+ | -] termo|
           expressão simples (or | + | -) termo
termo=factor |
      termo (and | * | div | mod | /) factor
factor=constante sem sinal |
      identificador-limite |
      variável |
      designador de função
      construtor de conjunto |
      (expressão) |
      (expressão) |
constante sem sinal=identificador de constante |
                    constante inteira sem sinal |

```

constante real sem sinal |
 constante-caracter |
 constante-cadeia |
 vazia
 construtor de conjunto=`[]` |
 `[ { designador de membro*, } ]`
 designador de membro=expressão |
 expressão..expressão

Como está implícito na sintaxe, os operadores possuem as seguintes precedências:

not (maior)
*and * div mod*
or+ -
 = < > < < = > = > *in* (menor)

e os operadores com a mesma precedência associam-se a partir da esquerda.

Os parêntesis negam em todos os casos as precedências dos operadores.

Se o tipo *S* é parte do tipo *T*, qualquer factor do tipo *S* é tratado como se fosse do tipo *T*. Do mesmo modo, qualquer factor do tipo *set of* (conjunto de) *S* é tratado como se fosse do tipo *set of T*, e qualquer factor do tipo *packed set of* (conjunto compacto de) *S* é tratado como se fosse do tipo *packed set of T*.

Todas as expressões num construtor de conjunto devem pertencer ao mesmo tipo ordinal *T*, e o próprio construtor de conjunto é do tipo *set of T* ou *packed set of T* (dependendo do contexto).

Semântica

A ordem de avaliação dos operadores é sempre dependente da aplicação. Se qualquer operando possui um valor indefinido, o programa falha.

Os significados dos operadores podem ser encontrados nas tabelas 2, 3, 5, 7, 8, 11 e 13.

O valor de `[]` é o conjunto vazio. O construtor de conjunt

`[M1,...,Mn]` possui o mesmo valor da união (`[M1] +...+ [Mn]`). O valor de `[e]` é o conjunto cujo único membro é o valor da expressão *e*. O valor de `[e1..e2]` é o conjunto cujos membros são os valores na gama entre *e₁* e *e₂* inclusive, ou o conjunto vazio se *e₁* > *e₂*.

Exemplos

Expressões booleanas:

- (a) soma > 0
- (b) caracter em ['a'..'z']
- (c) (balanço ≤ 0) e não excedido
- (d) *abs*(x-y) ≤ 1e-10
- (e) ref<> nula
- (f) nome='W. Shakespeare '
- (g) NomesOrdens- ['A'..'Z']<>[]

Expressões de caracteres:

- (h) *chr*(ord('a')-ord('A')+inicial)

Expressões de inteiros:

- (i) soma+somaneg
- (j) (hoje.ano-data.ano)*12+
(hoje.mês-data.mês)
- (k) i mod N+1

Outras expressões ordinais:

- (l) *succ*(hoje.mês)

Expressões reais:

- (m) *sqr*(*sqr*(x)+*sqr*(y))
- (n) *sin*(2*pi*j)
- (o) C1.partereal*C2.partereal-C1.parteimag*C2.parteimag

Expressões de conjuntos:

- (p) NomesOrdens*['A','E','I','O','U'] + ['Y']
- (q) ['A'..'inicial]

VARIÁVEIS

Utilização

As variáveis, quer se trate de variáveis completas quer de componentes de estruturas de dados, possuem valores que podem ser usados ou alterados.

Sintaxe

```
variável=variável completa |
    variável referenciada |
    variável indexada |
    designador de campo |
    variável-buffer
variável completa=variável-identificador
variável referenciada=variável-pointer ↑
variável indexada=variável-quadro
    [ { expressão-índice*, } ]
expressão-índice=expressão
designador de campo=[variável-registo.] identificador de
campo
variável-buffer=variável de ficheiro ↑
```

Uma variável de ficheiro, de registo, de quadro ou *pointer* é uma variável do tipo indicado.

O tipo da variável referenciada $P \uparrow$ é o tipo-domínio de P .

O tipo da variável indexada $A[i]$ é o tipo-componente de A ; a expressão-índice i deve ser compatível em termos de atribuição com o tipo-índice de A .

O tipo do designador de campo $R.F$ é o tipo do identificador de campo F do tipo registo de R . O designador de campo $R.F$ só pode ser abreviado para F no interior do bloco *with* S de uma declaração *with* R *do* S .

O tipo da variável-buffer $F \uparrow$ é o tipo-componente de F .

Semântica

A variável referenciada $P \uparrow$ denota a variável anónima de criação dinâmica à qual se refere o valor de P . Se o valor de P é nulo ou indefinido, o programa falha.

A variável indexada $A[i]$ denota o componente da variável-quadro A cujo índice é o valor de i . Se o valor de i sai fora da gama do tipo-índice de A , o programa falha.

O designador de campo $R.F$ denota o campo F da variável de registo R . A variável-buffer $\uparrow$ denota o *buffer* de ficheiro associado à variável de ficheiro F . A variável indexada $A[i_1, \dots, i_n]$ é uma abreviatura para a variável indexada $A[i] \dots [i_n]$.

Exemplos

- (a) $\text{ref } \uparrow$
- (b) $\text{ref } \uparrow.\text{seguinte } \uparrow$
- (c) $A[i] \{ \dots \text{ a } i\text{ésima linha de } A \}$
- (d) $A[i,j] \{ \dots \text{equivalente a } A[i][j] \}$
- (e) $\text{TabelaMeses}[\text{pred(mês)}]$
- (f) $\text{AntFicheiroContas } \uparrow.\text{possuidor}[1]$
- (g) hoje.ano
- (h) $\text{AntFicheiroContas } \uparrow.\text{conta}$
- (i) $\text{TabelaMeses}[\text{Jan}].\text{HorasSol}$
- (j) $\text{FicheiroTransacções } \uparrow.\text{data.ano}$
- (k) $\text{ref } \uparrow.\text{nome}$
- (l) $\text{NovoFicheiroContas } \uparrow$

PROCEDIMENTOS

Utilização

Um procedimento (*procedure*) é um bloco nomeado, possivelmente com parâmetros, que pode ser invocado para alterar o estado do processamento, por exemplo para alterar valores de variáveis, executar entradas/saídas, etc. Os procedimentos podem ser considerados como acréscimo ao repertório de instruções da Pascal. Um procedimento pode invocar-se a si mesmo.

Sintaxe

```

declaração-procedimento=
    nome de procedimento;
    (bloco-procedimento | ordem) |
    identificação-procedimento;
    bloco-procedimento
nome de procedimento=
    parte de parâmetros formais do identificador de proce-
dimento
identificação de procedimento=

```

```

procedimento identificador-procedimento
bloco de procedimento=bloco
instrução de procedimento=
    parte de parâmetros em uso do identificador de proce-
dimento

```

O nome de procedimento *procedure* I FPP define I como um identificador de procedimento no interior do bloco mais pequeno que o contém. O procedimento I consiste na parte de parâmetros formais FPP e num bloco de procedimento PB.

(1) O procedimento pode ser declarado numa única declaração de procedimento, com a forma:

```

procedure I FPP;
PB

```

(2) A ordem “standard” *forward* permite declarar PB numa declaração de procedimento separada (mas na mesma parte de função e procedimento):

```

procedure I FPP; { define I e FPP }
forward;

```

```

. . . . .
procedure I; { identificação-procedimento de I }
PB

```

Os procedimentos e funções intervenientes podem invocar o procedimento I.

(3) A ordem vulgar mas não “standard” *external* (ou *extern*) especifica que PB seja declarada no exterior do programa Pascal:

```

procedure I FPP;
external

```

Semântica

A declaração-procedimento I APP é executada do seguinte modo:

(1) A parte de parâmetros em uso APP é relacionada com a parte de parâmetros formais do procedimento I;

(2) É executado o bloco de procedimento de I.

Exemplos

Declarações de procedimentos e instruções de procedimentos:

```
(a) procedure EscreverData (data:Datas);  
    início  
    with data do  
        write (dia:2,'/',ord(mês)+1:2,'',(ano mod 100):2)  
    fim { EscreverData }
```

```
.....  
EscreverData (FicheiroTransacções ↑ .data)
```

```
(b) procedure LerNome (var Nome:CadeiasNome);  
    var col:1..16;  
    início  
    for col:=1 to 16 do  
        if not eoln then  
            read (nome[col])  
        else  
            nome[col]:=em branco  
    fim LerNome  
    { LerNome } (nome)
```

```
(c) procedure EscreverOctal (n:inteiro);  
    início  
    if n >= 8 then escreverOctal (n div 8);  
    write ((n mod 8):1)  
    fim { escreverOctal }  
    .....  
    escreverOctal (soma)
```

34

FUNÇÕES

Utilização

Uma função é um bloco nomeado, possivelmente com parâmetros, que pode ser invocado para produzir um valor. Uma função pode igualmente alterar o estado do processamento, mas estes efeitos laterais devem ser normalmente evitados. As funções podem ser consideradas como acrescentos ao repertório PASCAL de operadores. Uma função pode invocar-se a si mesma.

Sintaxe

Declaração de função=

nome de função;
(bloco-função|ordem) |
identificador de função;
bloco-função

Nome de função=

parte de parâmetros formais do identificador de função:
tipo-resultado

tipo-resultado=identificador-tipo
identificação de função=identificador-função

bloco-função = bloco
 declaração de atribuição de função=identificador de função:=expressão
 designador de função=parte de parâmetros em uso do identificador de função

O tipo-resultado pode não ser estruturado.

O nome de função *function* I FPP:T define I como um identificador de função, de tipo T, no interior do bloco mais pequeno que a contém.

A função I consiste na parte de parâmetros formais FPP e num bloco-função FB.

(1) A função pode ser declarada por uma única declaração de função, com a forma:

```
function I FPP:T;
  FB
```

(2) A ordem “standard” *forward* permite declarar FB numa declaração de função separada (mas na mesma parte de função e procedimento):

```
function I FPP:T; { define I,FPP e T }
  forward;
  .....
function I; { identificador de função de I }
  FB
```

Os procedimentos e funções intervenientes podem invocar a função I.

(3) A ordem vulgar mas não “standard” *external* (ou *extern*) especifica que FB é declarada no exterior do programa Pascal:

```
function I FPP:T;
  external
```

Semântica

O designador de função I APP é avaliado do seguinte modo:

- (1) A parte de parâmetros em uso é relacionada com a parte de parâmetros formais da função I;
- (2) É executado o bloco-função I;

(3) O último valor atribuído numa declaração de atribuição de função I:=E durante o passo 2 transforma-se no valor do designador de função. Se não foi executada qualquer declaração de atribuição de função, o valor do designador de função é indefinido.

Exemplos

Declarações e instruções de função contendo designadores de função:

```
(a) function mêsseguinte (mês:Meses):Meses;
  início
    if mês <> Dez then
      mêsseguinte:=succ(mês)
    else
      mêsseguinte:=Jan
  fim { mêsseguinte }
```

```
.....
mês:=mêsseguinte(hoje.mês)
(b) function anobissexto(ano:inteiro):booleana;
  { considera ano > 1752 }
  início
    anobissexto:=(ano mod 4=0) and
      ((ano mod 100<>0) or
      (ano mod 400=0))
  fim { anobissexto }
  .....
  if anobissexto (succ(hoje.ano)) then...
```

```
(c) function Adaptação (nomedado:CadeiasNome;
  { in } listadada:RefsListaNomes):RefsListaNomes;
  var ref:RefsListaNomes;
  adaptadas:booleano;
  início
    ref:=listadada; adaptada:=falsa;
    while not adaptada and (ref<>nula) do
      with ref ↑ do
        if nome=nomedado then
```

```

        adaptada:=verdadeiro
    else
        ref:=seguinte;
    Adaptação:=ref { nula se não adaptada }
    fim { Adaptação }
    .....
    ref:=Adaptação ('Niklaus Wirth',lista)
(d) function Ackermann (m,n:inteiro):inteiro;
. início
    if m=0 then
        Ackermann:=n+1
        Ackermann:=Ackermann(m-1,1)
    else if n=0 then
        Ackermann:=Ackermann(m-1,1)
    else
        Ackermann:=Ackermann(m-1,Ackermann
            (m,n-1))
    fim { Ackermann }
    .....
    soma:=Ackermann (2,i)

```

35

PARÂMETROS

Utilização

Permitir que uma função ou procedimento actue de modo diferente sobre diferentes objectos. Os parâmetros *em uso* são os objectos dados como operandos quando o procedimento ou função é invocado. Os parâmetros *formais* são os nomes pelos quais estes objectos são conhecidos no interior do procedimento ou função. Os parâmetros formais são classificados como parâmetros-valor, parâmetros-variáveis, parâmetros de procedimento ou parâmetros de função conforme a natureza dos objectos que representam.

Sintaxe

```

parte de parâmetros formais=
    [( { especificação de parâmetro formal*; } )]
especificação de parâmetro formal=
    especificação de parâmetro-valor |
    especificação de parâmetro-variável |
    especificação de parâmetro de procedimento |
    especificação de parâmetro de função
parte de parâmetros em uso = [( { parâmetro em uso*, } )]

```

parâmetro em uso = parâmetro-valor em uso |
 parâmetro-variável em uso |
 parâmetro de procedimento em uso |
 parâmetro de função em uso

Numa declaração de procedimento ou designador de função I APP, a parte de parâmetros em uso APP deve conter exactamente um parâmetro em uso por cada parâmetro formal do procedimento ou da função I. A correspondência entre parâmetros em uso e parâmetros formais é posicional.

Semântica

Cada parâmetro em uso é relacionado com o parâmetro formal correspondente à entrada do procedimento ou função. A ordem de relacionamento dos parâmetros depende da aplicação. Os parâmetros formais só existem durante a execução do procedimento ou da função.

36

PARÂMETROS — VALOR

Utilização

Permitir a um procedimento ou função utilizar (mas não alterar) os valores dos seus parâmetros em uso.

Sintaxe

especificação do parâmetro-valor=
 { identificador*, } :identificador de tipo
 parâmetro-valor em uso=expressão

A especificação do parâmetro-valor

$I_1, \dots, I_n : T$

especifica os identificadores $I_1, \dots, I_n$ como parâmetros-valor, de tipo T, do procedimento ou função. $I_1, \dots, I_n$ são identificadores-variáveis no interior do bloco-procedimento ou bloco-função.

O parâmetro-valor em uso correspondente a cada um destes parâmetros-valor deve ser uma expressão compatível em termos de atribuição com T (segue-se daqui que T não pode ser um tipo ficheiro nem um tipo estruturado com um componente de ficheiro).

Um parâmetro-valor é relacionado inicializando-o para o valor do parâmetro-valor em uso correspondente ao entrar no procedimento ou função. Se o valor do parâmetro-valor em uso estiver fora da gama aceite, o programa falha.

Exemplos

```
(a) function intlog2 (n:inteiro):inteiro;
    var I:inteiro;
    início
    I:=0;
    while n > 1 do
        início n:=n div 2;I:=I+1 fim;
    intlog2:=I
    fim { intlog2 }
.....
i:=intlog2(j) { ...não altera j }
```

37

PARÂMETROS – VARIÁVEL

Utilização

Permitir a um procedimento ou função actuar sobre variáveis fornecidas como parâmetros em uso.

Sintaxe

especificação do parâmetro-variável=
var { identificador*, } :identificador-tipo |
especificação de conformação de quadro
parâmetro-variável em uso=variável
A especificação do parâmetro-variável
var $I_1, \dots, I_n$:T

onde T é um identificador de tipo, especifica os identificadores $I_1, \dots, I_n$ como parâmetros-variável, de tipo T, do procedimento ou função.

$I_1, \dots, I_n$ são identificadores de variável no interior do bloco de procedimento ou função.

O parâmetro-variável em uso correspondente a cada um destes parâmetros-variável deve ser uma variável do mesmo tipo T. Não deve ser um componente imediato de uma estrutura compacta, nem um campo-limite de um registo.

Os parâmetros de conformação de quadro são descritos no capítulo seguinte.

Semântica

Um parâmetro-variável é relacionado fazendo-o representar a *identidade* do parâmetro-variável em uso correspondente, que é estabelecida (por exemplo avaliando quaisquer expressões-índice) ao entrar no procedimento ou função. Qualquer acesso ao parâmetro-variável é um acesso indirecto ao correspondente parâmetro-variável em uso.

Exemplos

```
(a) procedure incremento (var n:inteiro);  
  início  
    n:=n+1  
  fim { incremento }  
  .....  
  incremento (soma)
```

PARÂMETROS DE CONFORMAÇÃO DE QUADRO

Utilização

Permitir que um procedimento ou função actue sobre variáveis-quadro de diferentes dimensões em diferentes processamentos.

Sintaxe

```
especificação de conformação de quadro=  
  var { identificador*, } :esquema de conformação de  
quadro  
  esquema de conformação de quadro=  
    array [ { especificação de limites*; } ] of  
      (identificador-tipo | esquema de conformação de  
quadro)  
  especificação de limites=  
    identificador..identificador:tipo-limites  
  tipo-limites=identificador de tipo
```

Cada tipo-limites deve ser ordinal. A especificação de limites L..U:T define os identificadores L e U como identificadores de limite, de tipo T, no interior do bloco de procedimento ou função.

A especificação de conformação de quadro

var $I_1, \dots, I_n$:

array [$L_1..U_1:T_1$]*of*...*array* [$L_m..U_m:T_m$]*of* T

especifica os identificadores $I_1, \dots, I_n$ como parâmetros-variável do procedimento ou função. $I_1, \dots, I_n$ são identificadores de variável no interior do bloco de procedimento ou função.

Os parâmetros-variável em uso correspondentes a $I_1, \dots, I_n$ devem ser todos variáveis de quadro do mesmo tipo, T_a . Por outro lado, T_a deve ser *conformável* com o esquema de conformação de quadro, ou seja, T_a deve ter a forma

array [S_i]*of*...*array* [S_m]*of* T

onde cada tipo-índice S_i é compatível com o correspondente tipo-limites T_i , e onde a gama de valores de S_i se encontra no interior da gama de valores de T_i .

O tipo dos identificadores de variável $I_1, \dots, I_n$ é o próprio T_a , o que significa que esse tipo é flexível ao ponto de cada tipo-índice S_i de T_a poder ser qualquer subconjunto do correspondente tipo-limites T_i .

Semântica

Os parâmetros de conformação de quadro são relacionados do mesmo modo que os outros parâmetros-variável (ver capítulo anterior). Além disso, cada par de identificadores-limite L_i e U_i representa os valores mínimo e máximo do *ínésimo* tipo-índice S_i do tipo comum dos parâmetros-variável em uso correspondentes, T_a . Se estes valores mínimo e máximo estiverem fora da gama de T_i , o programa falha.

Abreviaturas

O esquema de conformação de quadro

array [$L_1..U_1:T_1; \dots; L_m..U_m:T_m$]*of* C

é uma abreviatura do esquema de conformação de quadro

array [$L_1..U_1:T_1$]*of*...*array* [$L_m..U_m:T_m$]*of* C

(onde C pode ser um identificador de tipo ou um esquema de

conformação de quadro).

Exemplos

(a) *procedure* SomaMatrizes

(*var* Matriz1, Matriz2, Soma:

array [$I1..u1$:inteiro;

$I2..u2$:inteiro] *of* real);

var linha, col:inteiro;

início

for linha:=I1 *to* u1 *do*

for col:=I2 *to* u2 *do*

Soma[linha,col]:=

Matriz1[linha,col]+Matriz2[linha,col]

fim { SomaMatrizes }

.....

SomaMatrizes (A,B,S) { processa $S_{ij}=A_{ij}+B_{ij}$ *for*
 $i=1..N, j=1..N$ }

PARÂMETROS DE PROCEDIMENTO

Utilização

Permitir a um procedimento ou função invocar um procedimento dado como um parâmetro em uso.

Sintaxe

especificação dos parâmetros de procedimento =
 parte de parâmetros formais simulados do identificador-*procedure*
 parte de parâmetros formais simulados = parte de parâmetros formais
 parâmetros de procedimento em uso = identificador-*procedure*

A especificação de parâmetros de procedimento *procedure* I DFPP especifica o identificador I como parâmetro de procedimento da *procedure* ou *function*. I é um identificador-*procedure* no interior do bloco de *procedure* ou função.

Os identificadores de parâmetros formais definidos em DFPP não têm qualquer significado.

O parâmetro de procedimento em uso correspondente a I deve ser um identificador-*procedure* P cuja parte de parâmetros

formais seja idêntica a DFPP (excepto que os identificadores de parâmetros formais P's podem diferir dos de DFPP). P não necessita de ser uma *procedure* previamente declarada.

Semântica

O parâmetro de procedimento I é relacionado com o correspondente parâmetro de procedimento em uso P levando I a denotar a *procedure* P. Nestas condições, qualquer declaração de procedimento invocando I tem como efeito invocar P.

Exemplos

```
(a) procedure ListaComparação
    (      listadada:RefsListaNomes;
    { aplicando } procedure p (ref:RefsListaNomes));
    var refnodo:RefsListaNomes;
    início
    refnodo:=listadada;
    while refnodo <> nula do
        início
        p(refnodo);
        refnodo:=refnodo ↑ .seguinte
        fim
    fim { ListaComparação }
    .....
procedure EscreverNome (refnodo:RefsListaNomes);
    início
    writeln (refnodo ↑ .nome)
    fim { EscreverNome }
    .....
    ListaComparação(Lista,EscreverNome) { escreve todos os nomes da Lista }
```

PARÂMETROS DE FUNÇÃO

Utilização

Permitir a um procedimento ou função invocar uma função apresentada como parâmetro em uso.

Sintaxe

especificação de parâmetro funcional=

parte de parâmetros formais simulados do identificador-função:tipo-resultado

parte de parâmetros formais simulados=parte de parâmetros formais

parâmetros de função em uso=identificador de função

A especificação de parâmetros funcionais *function* I DFPP:T especifica o identificador I como parâmetro funcional, de tipo T, do procedimento ou função. I é um identificador-função no interior do bloco de procedimento ou função.

Os identificadores de parâmetros formais definidos em DFPP não têm qualquer significado.

O parâmetro funcional em uso correspondente a I deve ser um identificador de função F cujo tipo resultado seja igual a T e cuja parte de parâmetros formais seja idêntica a DFPP (excepto no facto de os identificadores de parâmetros formais

de F poderem diferir dos de DFPP). F pode não ser uma função previamente declarada.

Semântica

O Parâmetro funcional I é relacionado com o correspondente parâmetro funcional em uso levando I a denotar a função F. Assim, qualquer designador de função que invoque I tem como efeito invocar F.

Exemplos

(a) *procedure* integrar

```
(      função F(x:real):real;
  { entre }   limiteinf,limitesup:real;
  { dando }   far integral:real;

  { avalia aproximadamente o integral simples de F(x)
    entre os limites dados, usando a decomposição de áreas
    com n intervalos }
  const n=8;
  var intervalo,soma:real;
      i:1..n;
  início
  intervalo:=(limitesup–limiteinf)/n;
  soma:=(F(limitinf)+F(limitesup))/2;
  for i:=1 to n-1 do
      soma:=soma+F(limitinf+i*intervalo);
  integral:=soma*intervalo
  fim { integrar }
  .....
  função tan(A:real):real;
  início
  tan:=sin(A)/cos(A)
  fim { tan }
  .....
  integrar (tan, { entre } 0,pi*x, { dando } y )
```

PROGRAMAS

Sintaxe

```

Programa=título de programa;
        bloco de programa
Título de programa=
        identificador de programa [( { parâmetro-progra-
ma*, } )]

```

Parâmetro de programa=identificador-variável

Bloco de programa=bloco

O identificador de programa não tem significado no interior do programa.

Cada parâmetro de programa deve ser declarado na parte de declaração de variáveis do bloco de programa. No entanto, se *input* for um parâmetro de programa, a declaração-variável

input:texto

é automaticamente inserida aqui. Do mesmo modo, se *output* for um parâmetro de programa, a declaração-variável

output:texto

é automaticamente inserida aqui.

Semântica

Os parâmetros de programa representam objectos externos (normalmente ficheiros). Em particular, os parâmetros de programa *input* e *output* (quando presentes) representam um ficheiro standard de texto de entrada e um ficheiro standard de texto de saída, respectivamente.

O programa é executado do seguinte modo:

(1) Os parâmetros de programa são relacionados com os correspondentes objectos externos (o modo como isto é feito é definido pela aplicação):

(2) se *input* for um parâmetro de programa, é executada a instrução *reset (input)*;

(3) se *output* for um parâmetro do programa, é executada a instrução *rewrite (output)*;

(4) é executado o bloco de programa.

Exemplos

(a) programa CópiaFicheiro (antficheiro, novoficheiro);
var

antficheiro, novoficheiro:ficheiro de inteiros;
{usado para quaisquer ficheiros}

início

reset(antficheiro); rewrite (novoficheiro);

while not eof (antficheiro) do

início

novoficheiro ↑ :=antficheiro ↑ ;

put (novoficheiro);

get (antficheiro)

fim

fim.

(b) programa CópiaFicheiroTexto (antficheiro, novoficheiro);

var

```

    antificheiro, novoficheiro:texto;
procedure CópiaUmaLinha;
    var
        caracter:char;
    início
        while not eoln (antificheiro) do
            início
                read (antificheiro, caracter);
                write (antificheiro, caracter)
            fim;
            writeln (novoficheiro)
        fim { CópiaUmaLinha } ;
    início { CópiaFicheiroTexto }
    reset (antificheiro); newwrite (novoficheiro);
    while not eof(antificheiro) do
        início
            CópiaUmaLinha;
            readln (antificheiro)
        fim
    fim { CópiaFicheiroTexto }
(c) programa MisturarFicheiros (Actualizações, AntMaster,
    NovoMaster);
    { mistura Actualizações e AntMaster, considerados am-
    bos ordenados por chave crescente. Os registos de Ac-
    tualização têm maior precedência quando os campos-cha-
    ve são iguais. Os registos misturados são passados para
    NovoMaster. }
    var
        AntMaster, Actualizações,
        NovoMaster:ficheiro de registos
            chave: inteiro;
            { + quaisquer outros campos }
        fim;
    { usado para chaves de qualquer tipo simples ou de
    cadeia }
    início

```

```

    reset (Actualizações);
    reset (AntMaster);
    rewrite (NovoMaster);
    while not (eof(Actualizações) or eof(AntMaster)) do
        início
            { usar buffers-ficheiro para previsão... }
            if Actualizações ↑ .chave > AntMaster ↑ .chave then
                início NovoMaster ↑ := AntMaster ↑ ;get (Ant-
                Master)
                fim;
            else
                início NovoMaster ↑ := Actualizações ↑ ;get
                (Actualizações) fim;
                put (NovoMaster)
                fim;
            while not eof (AntMaster) do
                início
                    NovoMaster ↑ := AntMaster ↑ ;get (AntMaster);
                    put(NovoMaster)
                fim;
            while not eof (Actualizações) do
                início
                    NovoMaster ↑ := Actualizações ↑ ;get (Actualiza-
                    ções);
                    put (NovoMaster)
                fim
            fim.

```

ÂMBITO DAS DEFINIÇÕES

Todos os identificadores usados num programa devem ser definidos. Nenhum identificador pode ser definido mais do que uma vez para o mesmo bloco.

O âmbito de uma definição do identificador I é o bloco B para o qual I é definido, incluindo todos os blocos encerrados em B, mas excluindo qualquer bloco interior a este para o qual exista outra definição de I. Assim, os âmbitos de duas definições de I não se sobrepõem, desde que se encontrem em blocos diferentes.

No interior do âmbito de uma definição de I, todas as outras ocorrências de I correspondem a esta definição e devem segui-la textualmente. Nenhuma ocorrência de I no exterior deste âmbito corresponde a esta definição.

Em tudo o que se refira ao âmbito das definições:

- (1) os identificadores de constante, de tipo, de função e de procedimento previamente definidos são considerados definidos num bloco imaginário encerrando todo o programa;
- (2) os identificadores de parâmetros formais são considerados como possuindo definições no início do bloco de procedimento ou função;

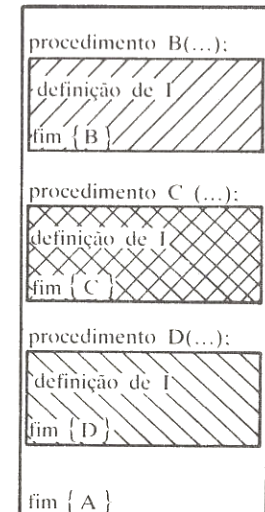
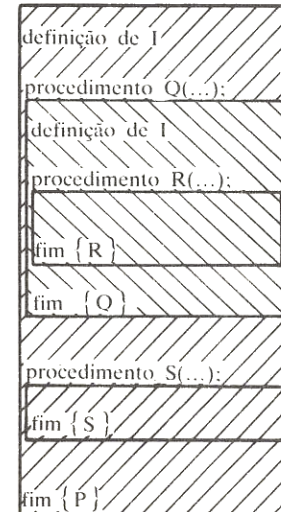
- (3) na declaração *with V do S*, todos os identificadores de campo da variável-registo V são considerados como definidos no início do bloco-*with S*;
- (4) uma descrição de registo actua como um bloco no que se refere a decidir o âmbito dos seus identificadores de campos.

Excepções a estas regras:

- (1) num designador de campo R.F, F deve ser um identificador de campo do tipo da variável-registo R;
- (2) um identificador-tipo T pode ocorrer numa descrição-*pointer* $\uparrow T$ em qualquer ponto da parte de definição de tipo que contenha a definição de T (não necessariamente a seguir a esta definição);
- (3) os identificadores de variáveis definidos no bloco-programa podem ser listados como parâmetros do programa.

Um identificador definido num bloco é considerado *local* a esse bloco, dado que não pode ser usado no exterior deste. Um identificador declarado no bloco-programa é considerado *global*, dado que pode ser usado em qualquer ponto do programa.

EXEMPLOS



Nos diagramas da página anterior os blocos estão encerrados em caixas, e os âmbitos das definições de I são distinguidos pelo sombreado.

- (a) programa P (...);
(figura 2 A)
- (b) programa A (...);
(figura 2 B)

ÍNDICE

1	Como utilizar este guia de bolso	7
2	Vocabulário: separadores	9
3	Vocabulário: termos fixos	11
4	Tipos e operações	14
5	Tipos ordinais	19
6	O tipo Booleano	22
7	O tipo character	24
8	O tipo inteiro	26
9	O tipo real	28
10	Tipos de indicadores	30
11	Tipos de quadros	33
12	Tipos de cadeias	36
13	Tipos de registos	38
14	Tipos de conjuntos	42
15	Tipos de ficheiros (incluindo entradas/saídas não textuais)	44
16	O tipo texto (incluindo entradas/saídas textuais) ..	48
17	Blocos	52
18	Definições de constantes	54
19	Definições de tipo	56
20	Declarações de variáveis	58
21	Instruções	60
22	Declarações de atribuição	62
23	Instruções compostas	64
24	Instruções IF	66

25	Declarações de caso	68
26	Instruções WHILE	70
27	Instruções REPEAT	72
28	Instruções FOR	74
29	Instruções WITH	76
30	Instruções e etiquetas GOTO	78
31	Expressões	81
32	Variáveis	84
33	Procedimentos	86
34	Funções	89
35	Parâmetros	93
36	Parâmetros – valor	95
37	Parâmetros – variável	97
38	Parâmetros de conformação de quadro	99
39	Parâmetros de procedimento	102
40	Parâmetros de função	104
41	Programas	106
42	Âmbito das definições	110

Fotocomposto por Nova Força
e acabou de se imprimir em 1985
por Tipografia Nunes, Lda. — Porto
para a EDITORIAL PRESENÇA. LDA.