Algumas notas sobre PASCAL

PARTE I

Anjolina Grisi de Oliveira

1 Um programa em PASCAL

Esse texto inclui algumas notas sobre a linguagem PASCAL. As diferenças existentes entre os diversos compiladores não serão consideradas. Foram consultados os livros [CooCla85], [Ziv99] e [Asc99].

A estrutura de um programa em PASCAL é basicamente definida como a seguir:

Program	cabeçalho do programa
const	definição de constantes
type	definição dos tipos dos dados
var	declaração das variáveis
procedure e/ou **function**	declaração dos procedimentos e funções
Begin	início do programa principal
corpo do programa	
End.	fim do programa principal

2 Tipos de dados

Na linguagem PASCAL as variáveis, constantes e funções possuem um tipo associado. Os tipos podem ser classificados como a seguir:

1. simples ou básico: *integer*. *char*, *boolean*, *real*, etc.
2. estruturados ou compostos: *array*, *record*. *file* e *sets*.
3. Apontadores ou ponteiros: um apontador é uma variável que contém o endereço de uma outra variável *alocada dinamicamente*.

Os tipos simples também são chamados tipos primitivos. Os tipos estruturados são construídos a partir dos tipos primitivos.

2.1 O tipo *array*

As variáveis simples fazem referência a apenas um único valor. Entretanto, as variáveis estruturadas, como por exemplo o tipo *array*, podem armazenar uma coleção de valores do mesmo tipo, por isso é chamado de um tipo de dado *homogêneo*.

Os arranjos (*arrays*) são estruturas n-dimensionais. Nesse texto, iremos abordar os arranjos unidimensionais, conhecidos como **vetores** e os arranjos bi-dimensionais, conhecidos como **matrizes**.

Vetores

Os vetores são arranjos unidimensionais. É uma sequencia de variáveis, todas do mesmo tipo, que são alocadas em posições contíguas na mémoria. A figura 1 ilustra um arranjo desse tipo.

Declarando vetores

```
var
   <identificador> : array[limiteinferior .. limitesuperior] of <tipo dos elementos>;
```

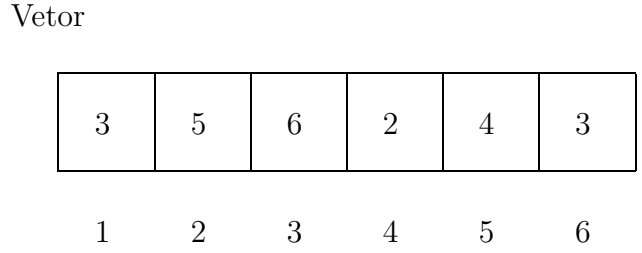

Figura 1: Vetor

Exemplo 1 *O seguinte programa em PASCAL ler um vetor de 6 posições inteiras e imprime seus elementos na ordem reversa.*

```
Program reverte;
var i: integer;
    vetor: array[1..6] of integer;
Begin
    for i:=1 to 6 do
        readln(vetor[i]);
    for i:=6 downto 1 to
        writeln{vetor[i]);
    readln;
End.
```

Atenção:

- para a diferença entre os elementos de um vetor e os índices; Na figura 1, o primeiro elemento é o número 3, ou seja, vetor[1] é 3. O índice é 1 e o conteúdo é 3.
- O *tamanho* de um vetor é o número de elementos. No caso do exemplo 1, o vetor possui 6 elementos;
- a declaração de um vetor inclui os seus limites: inferior e o superior e não o seu tamanho. No exemplo 1 os limite inferior é 1 e o limite superior é 6. Entretanto, poderiamos alterar a definição desse vetor:

```
var vetor: array[0..5] of integer;
```

Nesse caso, o vetor continua tendo 6 elementos, sendo que os limites são diferentes da primeira definição. Devem ser feitas alterações apropriadas nos limites dos comandos *for* também.

Podemos definir um vetor de uma forma diferente, usando a declaração *type*, conforme ilustrado a seguir.

```
type tipovetor = array[1..6] of integer;
var
   vetor : tipovetor;
```

Exemplo 2 *Os três arranjos definidos a seguir possuem 6 elementos:*

```
var
    vet1: array[2..7] of integer;
    vet2: array[-3..2] of char;
    vet3: array['A'..'F'] of real;
```

Atribuindo valores a um vetor

Considerando os vetores definidos no exemplo 2, podemos atribuir valores a eles da seuinte forma:

```
vet1[2] := 3;
vet2[0] := 'B';
vet3['A'] := 3.2;
```

Mais um exemplo

O seguinte programa, ilustrado no livro [CooCla85], mostra a importância do uso de vetores na programação.

Exemplo 3 *Suponha que um programa receba como entrada uma longa sequencia de números inteiros. Nós desejamos contar o número de vezes em que cada caracter, que é um dígito ocorre na sequencia.*

```
Program ContaDig;
type
     Totais = array['0'..'9'] of integer;
var
     Digitos: Totais;
     DigitoAtual: char;
     i, Quant: integer;
Begin
    for DigitoAtual:='0' to '9' do
        Digitos[DigitoAtual]:=0;
```

```
    writeln('Quantos digitos possui a sequencia?');
    readln(Quant);
    writeln('Entre com a sequencia de numeros');
    for i:=1 to Quant do
      begin
        readln(DigitoAtual);
        Digitos[DigitoAtual] := Digitos[DigitoAtual] + 1;
      end;
    writeln('Numero de vezes que cada digito ocorre na sequencia:');
    for DigitoAtual:='0' to '9' do
       writeln(DigitoAtual,'=',Digitos[DigitoAtual]);
    readln;
End.
```

Matrizes

Matrizes são arranjos bi-dimensionais.

Declarando matrizes

```
var
  <identificador>:array[l1..l2,c1..c2] of <tipo dos elementos>;
```

Onde l1 e l2 são os limites das linhas da matriz, enquanto que c1 e c2 são os limites das colunas.

Exemplo 4 *Exemplos de variáveis do tipo matriz:*

```
    var
       Matriz1: array[1..5,1..5] of integer;
       Matriz2: array[1..3,1..8] of char;
```

Atribuindo valores às matrizes

```
     Matriz1[1,2]:=0;
     Matriz2[3,8]:='A';
```

Exemplo 5 *O seguinte programa faz a leitura de uma matriz de números reais e imprime a soma dos elementos da linha e da coluna fornecidos pelo usuário.*

```
 Program ExMatriz;
 var
    Mat: array[1..4,1..3] of real;
    i,j,linha,coluna: integer;
    somanl,somac:real;
    DadosCorretos: boolean;
 Begin
    somal:=0.0; somac:=0.0;
    DadosCorretos := false;
```

```
    i:=1;
    while (i<=3) and (not DadosCorretos) do
      begin
        writeln('Entre com o numero da linha e da coluna a ser somada');
        readln(linha,coluna);
        if (linha < 1) or (linha > 4) or (coluna < 1) or (coluna >3) then
          begin
            writeln('dados incorretos');
            writeln('voce tem mais',3-i,'chances');
          end
        else
          DadosCorretos:=true;
        i:=i+1;
      end;
    if DadosCorretos then
      begin
        writeln('Entre com a matriz');
        for i:= 1 to 4 do
          for j:=1 to 3 do
            begin
              writeln('Elemento',i,' ',j);
              readln(Mat[i,j]);
            end;
        for j:=1 to 3 do
          somal := somal + Mat[linha,j];
        for i:=1 to 4 do
          somac := somac + Mat[i,coluna];
        writeln('soma dos elementos da linha ',linha,'=',somal);
        writeln('soma dos elementos da coluna ',coluna,'=',somac);
      end;
    readln;
  End.
```

2.2 O tipo *record*

O tipo *record* (registro) é um tipo de dado estruturado heterogêneo. Isso porque ele pode agrupar dados de tipos diferentes. O tipo registro é usado quando os dados agrupados têm alguma afinidade, ou seja, estão logicamente relacionados. Sua sintaxe no PASCAL é definida com a seguir:

```
type <nome do registro> = record
     nome campo1 : tipo do dado do campo1;
     nome campo2 : tipo do dado do campo2;
        .
        .
     nome campo-n: tipo de dado do campo-n;
```

```
    end;
```

Acesso a um campo do registro

Usa-se um mecanismo conhecido como *dot notation*:

```
      nomedoregistro.campo
```

Exemplo 6 *Ilustrando o uso de registros.*

```
type tipoplaca = record
       alfa : string[3];
       num : integer;
     end;

type tipocarro = record
       placa : tipoplaca;
       ano: integer;
     end;

var
   carro: tipocarro;

begin
   readln(carro)
   writeln('placa do carro lido =',carro.placa,' ano = ',carro.ano);
   writeln('parte alfa da placa =',carro.placa.alfa);
   writeln('parte numerica da placa =',carro.placa.num);
   readln;
end.
```

O comando with

Com esse comando é possível acessar os componentes de um registro utilizando-se apenas os nomes dos campos.

```
   with registro1, registro2, ..., registron do
     begin ... end;
```

Exemplo 7 *Os seguintes trechos de programa possuem o mesmo efeito.*

```
      with carro do
       begin
           ano = 2003;
           placa.alfa = 'KKI';
```

```
        placa.num = 3241;
    end;

   with carro,placa do
    begin
        ano = 2003;
        alfa = 'KKI';
        num = 3241;
    end;
```

Exemplo 8 *Esse exemplo [CooCla85] ilustra o uso de arrays de registros.*

```
type
   CoresPeca = (nenhuma,preta,branca);
   ValoresPeca = (vazio,rei,rainha,bispo,piao,torre,cavalo);
   quadrados = record
                 ocupado : boolean;
                 peca    : ValoresPeca;
                 CorPeca : CoresPeca;
               end;
   TabuleiroXadrez = array[1..8,1..8] of quadrados;

var
   UltimoMovimento, MovimentoAtual : quadrados;
   Tabuleiro : TabuleiroXadrez;
```

Assuma que o elemento 5,2 é um pião branco. Os comandos definidos a seguir movem o pião branco da posição 5,2 para a posição 5,4. Todas os blocos de comandos mostrados abaixo fazem a mesma tarefa:

```
BLOCO1:

    Tabuleiro[5,4] := Tabuleiro[5,2];

BLOCO2:

    Tabuleiro[5,4].ocupado := true;
    Tabuleiro[5,4].peca := piao;
    Tabuleiro[5,4].corpeca := branca;

BLOCO3:

    MovimentoAtual.ocupado := true;
    MovimentoAtual.peca := piao;
    MovimentoAtual.corpeca := branca;
    Tabuleiro[5,4] := MovimentoAtual;

BLOCO4:
```

```
with Tabuleiro[5,4] do
  begin
     ocupado := true;
     peca := piao;
     corpeca := branca;
  end;
```

Após a execução de um desses blocos, precisamos esvaziar a posição 5,2:

```
with Tabuleiro[5,2] do
 begin
     ocupado := false;
     peca := vazio;
     corpeca := nenhuma;
 end;
```

Referências

[Asc99] Ana Fernanda Gomes Ascencio. Lógica de programação com Pascal. Makron Books Ltda, São Paulo, Brasil. 1999.

[CooCla85] Doug Cooper and Michael Clancy. Oh! Pascal! W.W. Norton and Company, New York and London. Second Edition, 1985.

[VeSaAzFu83] . Veloso, C dos Santos, P. Azeredo and A. Furtado. Estrutura de Dados Editora Campus, Brasil, 15a edição, 1983.

[Ziv99] Nivio Ziviani. Projeto de Algoritmos com implementações em Pascal e C. Pioneira Informática, São Paulo, Brasil. Quarta edição, 1999.