Estruturas de Dados Homogêneas: Vetores e Matrizes

1. Introducão

Até o momento, sempre que foi preciso armazenar temporariamente dados, usamos as variáveis. Entretanto, variáveis somente podem armazenar um único valor por vez e isso torna seu uso limitado para resolver problemas que envolvam uma quantidade maior de dados. Logo, nesses casos, usam-se as estruturas de dados denominadas vetores e matrizes.

2. Vetores

Enquanto uma variável pode armazenar sempre um único valor, os vetores podem armazenar qualquer quantidade pré-estabelecida de valores. Encare um vetor como uma coleção de variáveis do mesmo tipo (homogêneo).

Sintaxe para declaração de Vetores

<nome do vetor> **: vetor[**<valor inicial> **. .** <valor final>**] de** <tipo>

Observando a sintaxe de declaração acima posta, é possível notar uma das vantagens no uso de vetores – a reserva de uma grande quantidade de espaços capazes de armazenarem dados de uma maneira sucinta.

Exemplos:

```
nomes : vetor[1. . 100] de caracter;
idades : vetor[1. . 100] de inteiro;
salarios: : vetor[1. . 100] de real;
```

No exemplo acima tem-se declarado 3 vetores respectivamente denominados nomes, idades, e salários (vetores normalmente são declarados no plural). Somados, os 3 vetores são capazes de armazenar ate 300 “informações” distintas em um determinado instante, o equivalente a 300 varáveis, porém só foi necessário 3 linhas para reservar todo esse espaço na memória. Na verdade, poderíamos ter uma quantidade ainda maior declarada desde de que aumentássemos o “valor final” na declaração de cada vetor de 100 para qualquer outro valor inteiro positivo. Note que o limite para esse número é a quantidade de memória disponível no computador e que, como o vetor é uma estrutura de dados estática (não aumenta ou diminui em tempo de execução), o número de elementos deve ser conhecido no momento da declaração.

Mas como manter todos esses dados organizados? Para isso, é necessário usar índices indicando as posições onde deseja-se armazenar uma determinada informação. Em cada posição do vetor é possível armazenar um único valor, mas atenção para usar índices que excedem o valor inicial e/ou final informados na declaração do vetor. Observe o exemplo:

```
Exemplos:

 nomes[1] <- " João Martins"
 idades [1]< - 30
 salarios[1]<- 1000

 leia(nomes[2])
 leia(idade[2])
 leia(salarios[2])
```

No exemplo acima, todos os dados referentes a "João Martins" estão armazenados nas posições de índice 1 dos respectivos vetores, nesse caso, a primeira posição de cada vetor. Já as informações armazenadas no índices 2 de cada vetor são informadas pelo usuário. Como visto, assim como nas variáveis, só é possível armazenar informações nos vetores usando comandos de atribuição ou comando de entrada.

O uso de índices para referenciar a posição em que se deseja armazenar ou recuperar uma informação possibilita o uso de estruturas de repetição para percorrer as posições dos vetores. Por exemplo, suponha que desejamos saber a soma de todos os salários armazenados nos vetor de mesmo nome. Podemos fazer isso de duas formas:

```
//opção 1 : Sem estrutura de repetição
Soma  <-salarios[1] + salarios[2] + salarios[3] + salarios[4]+ ... +
salários[100];

//opção 2: com estrutura de repetição
soma<-0
para i <- 1 ate 100 faca
  soma<- soma + salarios[i];
fimpara
```

Na primeira opção somamos todos os valores individualmente, como se fosse variáveis. Isso é muito trabalhoso além de dificultar a manutenção do programa, caso seja necessário aumentar o tamanho do vetor. A segunda opção é bem mais inteligente e, caso o tamanho do vetor mude, só precisamos aumentar o número de ciclos da estrutura de repetição.

Importante: Um erro muito comum ao se trabalhar com vetores é não referenciar a posição em que se deseja armazenar ou recuperar a informação. Exemplo:

Escreva (salarios) - Isso é errado, pois "salarios" é um vetor e não uma variável.

Escreva(salários[1]) - Ok..agora está correto!

3. Matrizes

Matrizes são vetores de vetores, ou seja, uma coleção de vetores que por sua vez é uma coleção de variáveis. Talvez a representação mais comum da matriz, seja uma tabela com linhas e colunas.

Observe as duas representações abaixo. Computacionalmente, as duas são equivalentes, entretanto a primeira representação enfoca a existência de um vetor de 4 elementos dentro de um outro vetor mais externo de 3 elementos, sendo cada um desses elementos em vetor per si. Esse tipo de matriz é dita bidimensional, pois existe a dimensão das linhas e a dimensão das colunas.

Linha 1, coluna 1	Linha 1, coluna 2	Linha 1, coluna 3	Linha 1, coluna 4
Linha 2, coluna 1	Linha 2, coluna 2	Linha 2, coluna 3	Linha 2, coluna 4
Linha 3, coluna 1	Linha 3, coluna 2	Linha 3, coluna 3	Linha 3, coluna 4

Matriz 1: Primeira representação

Linha 1, coluna 1	Linha 1, coluna 2	Linha 1, coluna 3	Linha 1, coluna 4
Linha 2, coluna 1	Linha 2, coluna 2	Linha 2, coluna 3	Linha 2, coluna 4
Linha 3, coluna 1	Linha 3, coluna 2	Linha 3, coluna 3	Linha 3, coluna 4

Matriz 2: Segunda Representação

Sintaxe para declaração de Matrizes

<nome da matriz> **: vetor[**<linha inicial> **. .** < linha final>,
<coluna inicial> **. .** < coluna final>**] de** <tipo>

Exemplos

```
//Salários de 100 funcionários em cada um dos 12 meses;
salariosMensais: vetor[1..100, 1..12] de real
```

No exemplo acima, a matriz foi declarada de forma que cada funcionário tenha seu salário mensal armazenado em cada uma das 12 colunas. Suponha que desejamos saber quanto em média se pagou por mês para os funcionários . Para tanto, precisamos somar os salários de todos os funcionários em determinado mês (coluna) e dividir pelo número de funcionário (linhas). Devemos faze isso para cada um dos 12 meses.

```
Para mes<- 1 ate 12 faca
  pagamentoMensal<- 0
  Para funcionário<- 1 ate 100 faca
       pagamentoMensal<- pagamentoMensal +
                                    salariosMensais[funcionário, mes]
   fimpara
   pagamentoMensal<- pagamentoMensal/100
    escreva( “O pagamento médio no mes”, mes, “ foi de R$”, pagamentoMensal)
fimpara
```

4. Nota sobre o Cadastro de Vetores e Matrizes

Vetores e matrizes permitem manipular grande quantidade de dados. Na prática, esses dados normalmente são obtidos de uma fonte externa, ou seja, não os lemos diretamente do usuários. Entretanto, se não o fazemos, não é possível testar se os algoritmos estão funcionando a contento. Logo, sugere-se o cadastro randômico dos elementos da matrizes que varia de acordo com a linguagem que estiver sendo usada na implementação.