Vetores e Matrizes

André Lawisch

Vetor

Até agora, todas as variáveis com que trabalhamos eram unitárias. Ou seja, sempre trabalhamos com elementos (valores) únicos. Porém, esta abordagem nem sempre é adequada para solucionar determinados problemas, principalmente quando precisamos lidar com muitos valores simultaneamente.

Vetor

Imaginemos a criação de mil variáveis para representar todas as notas dos alunos de uma escola. E a manipulação desses valores, como se daria?

Vetor

Este problema é solucionado com a utilização de **variáveis compostas ou indexadas**.

Estas variáveis representam um conjunto de valores e tem um nome, cada unidade deste conjunto representa uma variável única.

Vetor

Cada unidade será referenciada através de um índice para que possa ser diferenciada das demais unidades do conjunto. Essas variáveis compostas ou indexadas podem ser do mesmo tipo (**homogêneas**) ou de tipos diferentes (**heterogêneas**).

Vetor Unidimensional

As variáveis compostas ou indexadas que possui apenas um índice são chamadas de vetores ou variáveis compostas unidimensionais.

Declaração de um vetor:

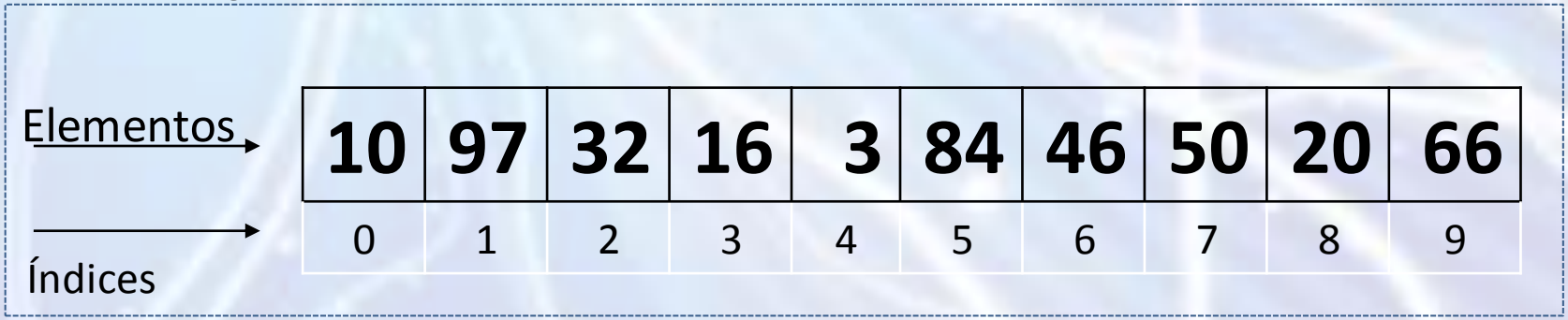
tipo nome_var[tamanho];

float n[10];

Vetor Unidimensional

O vetor (conjunto) **n** possui 10 elementos (variáveis) do tipo real e cada elemento é referenciado por um índice, como mostra a ilustração abaixo:

conjunto **n**



Vetor Unidimensional

Cada elemento do vetor **n** é manipulado como se fosse uma variável única, mas com índice.

Assim teríamos:

n [0] = 5;

n [1]= n[7] + 32;

5	**82**	**32**	**16**	**3**	**84**	**46**	**50**	**20**	**66**
0	1	2	3	4	5	6	7	8	90

Vetor Unidimensional

Um vetor não pode ser manipulado como uma única variável. Toda a manipulação de um vetor deve ser feita individualmente.

~~*escreva (n)*~~
~~*leia (n)*~~

forma errada de manipular um vetor

leia (n[2])
escreva (n[2])

forma correta de manipular um vetor

Vetor Unidimensional

Exemplo: Escreva um código que leia 50 valores e os escreva.

```
#include <stdio.h>
#include <stdlib.h>
int main(void)
{
    int a0,a1,a2,...,a49;
    scanf("%d"&a0);
    scanf("%d"&a1);
    ....
    scanf("%d",&a49);
    printf("%d",a0);
    printf("%d",a1);
    ....
    printf("%d",a49);
    system("pause");
}
```

Como este código trabalha com 50 valores, devemos criar 50 variáveis e manipulá-las de forma independente, ou seja, 50 linhas de leitura e mais 50 de escrita.

Vetor Unidimensional

Exemplo: Faça um algoritmo que leia 50 valores e os escreva.

```
#include <stdio.h>
#include <stdlib.h>
int main(void)
{
      int a[50];
      scanf("%d"&a[0]);
      scanf("%d"&a[1]);
      .....
      scanf("%d",&a[49]);
      printf("%d",a[0]);
      printf("%d",a[1]);
      ....
      printf("%d",a[49]);
      system("pause");
}
```

Para resolver o problema apontado na lâmina anterior, vamos utilizar um vetor. Analisando o código, percebemos que não houve alteração no tamanho do código.

Vetor Unidimensional

Recapitulando: vetor **a** com 50 posições

a[0] a[1] a[2] a[3] a[4] . . . a[49]

```
scanf("%d",&a[0]);
scanf("%d",&a[1]);
scanf("%d",&a[2]);
scanf("%",&a[3]);
...
scanf("%d",&a[49]);
```

Analise os comandos ao lado e responda: **o que é diferente em cada linha?**

Vetor Unidimensional

Recapitulando: vetor **a** com 50 posições

a[0] a[1] a[2] a[3] a[4] . . . a[49]

```
scanf("%d",&a[0]);
scanf("%d",&a[1]);
scanf("%d",&a[2]);
scanf("%d",&a[3]);
...
scanf("%d",&a[49]);
```

O que difere em cada linha é o valor do **índice**.

Considerando que o índice é uma expressão aritmética, podemos substituí-lo por uma variável (**scanf("%d"&a[0]);**) e esta linha ser colocada dentro de uma repetição que será executada 50 vezes.

Vetor Unidimensional

```
#include <stdio.h>
#include <stdlib.h>
int main(void)
{
    int a[50],i;
    for(i=0;i<50;i++)
    {
        scanf("%d",&a[i]);
    }
    for(i=0;i<50;i++)
    {
        printf("%d",a[i]);
    }
    system("pause");
}
```

Quando utilizamos repetições para variar o índice de um vetor, temos códigos menores, independente do tamanho do vetor

Vetor Unidimensional

Advertência

Não existe um comando para realizar a leitura, escrita, comparação dos valores de um vetor de uma única vez. Portanto, é obrigatório utilizar repetições para manipular um vetor.

```
for(i=0;i<50;i++)
{
    scanf("%d",&a[i]);
}
```

Não existe outra maneira

Vetor Unidimensional

Modelo de uma estrutura básica para manipular um vetor

```
algoritmo “!"
var
   x:vetor[1..20] de inteiro //declara vetor
   i:inteiro //declara variável para índice
inicio
   para i de 1 ate 20 faca // intervalo dos índices
      leia(x[i])  //instrução para executar
   fimpara
   para i de 1 ate 20 faca // intervalo dos índices
      escreva(x[i])  //instrução para executar
   fimpara
fimalgoritmo
```

Vetor Unidimensional

Exercício: Ler valores para um vetor X[20] e escreva somente os valores pares.

```
#include <stdio.h>
#include <stdlib.h>
int main(void)
{
   int a[20],i;
   for(i=0;i<20;i++)
   {
      scanf("%d",&a[i]);
   }
   for(i=0;i<20;i++)
   {
      if(!(a[i]&1))
            printf("%d",a[i]);
   }
   system("pause");
}
```

Vetor Bidimensional

Os vetores, ou variáveis compostas unidimensionais, têm como principal característica a necessidade de apenas um índice para endereçamento.

Uma estrutura que precise de mais de um índice será denominada estrutura composta multidimensional.

Vetor Bidimensional

As variáveis compostas multidimensionais mais utilizadas são as bidimensionais, também chamadas de matrizes.

Geralmente, utilizamos matrizes em situações que precisam de linhas e colunas para a identificação de elementos.

Matriz – X[6][6]

int X[6][6];

X_{00}	X_{01}	X_{02}	X_{03}	X_{04}	X_{05}
X_{10}	X_{11}	X_{12}	X_{13}	X_{14}	X_{15}
X_{20}	X_{21}	X_{22}	X_{23}	X_{24}	X_{25}
X_{30}	X_{31}	X_{32}	X_{33}	X_{34}	X_{35}
X_{40}	X_{41}	X_{42}	X_{43}	X_{44}	X_{45}
X_{50}	X_{51}	X_{52}	X_{53}	X_{54}	X_{55}

Os números subscritos indicam os índices dos elementos da matriz. O primeiro refere-se a linha e o segundo a coluna.

Matriz

```
#include <stdio.h>
#include <stdlib.h>
int main(void)
{
    int X[6][6],l,c;
    for(l=0;l<6;l++)
    {
        for(c=0;c<6;c++)
        {
            scanf("%d",&[l][c]);
        }
    }
    system("pause");
}
```

Agora, para manipular uma matriz, são necessários dois índices. O primeiro referencia a linha e o segundo, a coluna. Sempre nesta ordem.

Matriz

A atribuição é uma das formas de qualquer variável armazenar algum valor. Como não **operamos diretamente com a matriz**, somente seus elementos armazenam valores numa atribuição. Assim, é necessário a posição (linha e coluna) da matriz que devera receber o valor.

```
X[0][0] = 20;
X[3][2] = 2;
X[2][5] = X[1][4] + X[4][1];
```

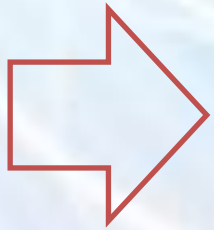
Matriz

```
int main(void)
{
    int X[10][10],l,c;
    for(l=0;l<10;l++)
    {
        for(c=0;c<10;c++)
        {
            if(l==c) X[l][c] = 1;
            else X[l][c] = 0;
        }
    }
    system("pause");
}
```

Matriz

Faça um programa que leia uma matriz X 2x5 de inteiros e imprima os elementos de X na ordem inversa.

x_{00}	x_{01}	x_{02}	x_{03}	x_{04}
x_{10}	x_{11}	x_{12}	x_{13}	x_{14}



x_{14}	x_{13}	x_{12}	x_{11}	x_{10}
x_{04}	x_{03}	x_{02}	x_{01}	x_{00}

Matriz

```
#include <stdio.h>
#include <stdlib.h>
int main(void)
{
    int x[2][5],l,c;
    for(l=0;l<2;l++)
    {
        for(c=0;c<5;c++)
        {
            scanf("%d",&x[l][c]);
        }
    }
    for(l=1;l>=0;l--)
    {
      for(c=4;c>=0;c--)
      {
            printf("%d",x[l][c]);
      }
    }
    system("pause");
}
```