Algoritmos e Lógica de Programação

PROF. NILTON

Aula de Hoje

- Vetores
- Matrizes
- Ordenação de Vetores
- Busca linear e binária

Vetores

Até o momento, todas as variáveis que utilizamos em nossos programas eram atômicas, ou seja, armazenavam apenas um valor de cada vez de acordo com seu tipo. Porém essa abordagem nem sempre é adequada para solucionarmos determinados problemas, principalmente quando precisamos administrar muitos valores simultaneamente.

Vetores

Vamos imaginar que em nosso programa iremos controlar as notas de vários alunos de uma sala de aula ou as vagas de um estacionamento rotativo de veículos, seria necessário a criação de tantas variáveis quanto o número de objetos a serem representados.

Essa quantidade de variáveis pode tornar a elaboração do algoritmo inviável. Imaginemos que será necessário a criação de mil variáveis para representar as notas de todos os alunos, ou quinhentas variáveis para representar as vagas do estacionamento. E a manipulação dessas variáveis, como seria feito?

Vetores

Tais problemas são tratados dentro da computação por meio da utilização de variáveis compostas. Essas variáveis podem representar uma coleção de dados do mesmo tipo.

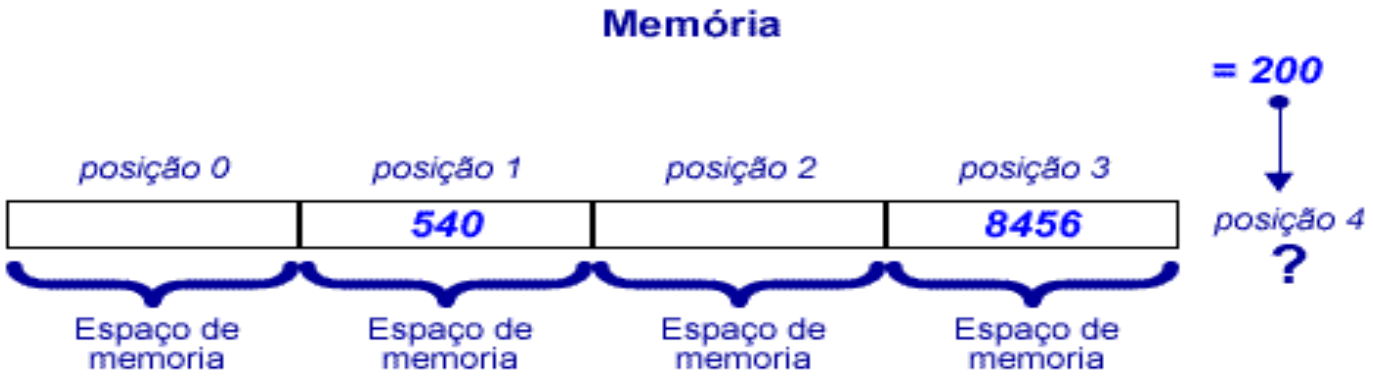
Podemos considerar tais variáveis como um subconjunto de um determinado tipo de dados.

Os vetores também são conhecidos como array.

Vetores

Outra característica essencial dos vetores é que seus elementos têm ORDINALIDADE, ou seja, possuem um ordem sequencial que identifica os elementos por sua posição. Assim podemos identificar o primeiro, o segundo, até o ultimo elemento do conjunto.

Podemos dizer que o vetor é uma variável indexada, pois cada posição é identificada por um índice.



Vetores

Vejamos a declaração de um vetor:

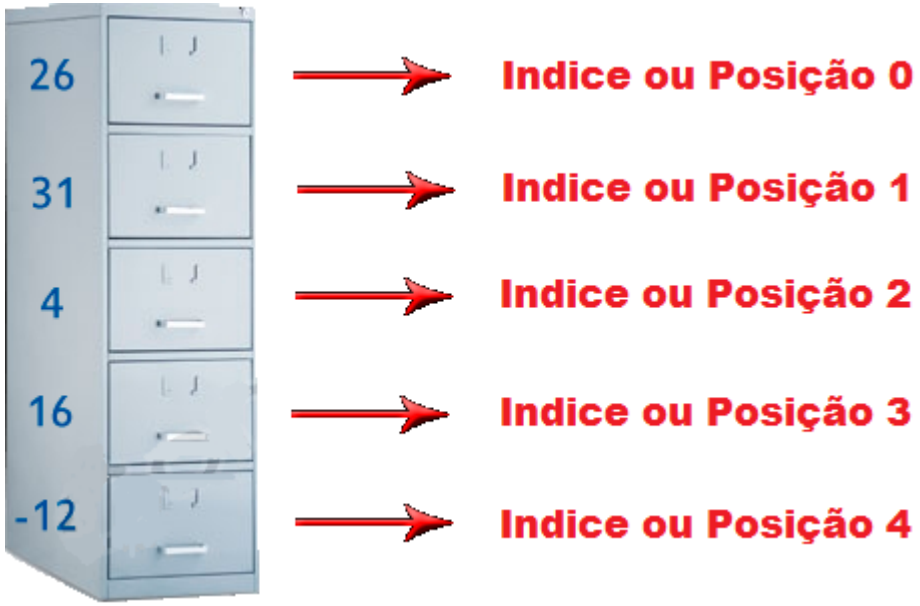
<tipo_do_dado> <nome_do_vetor> [n]; onde n é um número inteiro positivo que estabelece o tamanho do vetor.

Exemplo:

int notas[5] ;

Vetores

O acesso a esses elementos é feito pela sua posição dentro do vetor. No C/C++, o vetor é indexado a partir do índice 0 até o tamanho do vetor menos 1.



Vetores

Preenchendo um vetor de 4 posições:

```
int notas[4];
notas[0] = 7;
notas[1] = 5;
notas[2] = 9;
notas[3] = 8;

notas[4] = 3;  /* ERRADO – índice fora dos limites do vetor */
```

Vetores

Lendo informações de um vetor e atribuindo a uma variável comum:

Nota1 = notas[0];

Nota2 = notas[1];

Exibindo informações de um vetor no console:

*cout << "Primeira nota: " << **notas[0]**;*

Vetores de tamanho constante

Um vetor com tamanho constante pode ser iniciado no momento de sua declaração. Neste caso os valores iniciais devem ser indicados entre chaves {} e separados por virgula. Exemplos:

```
char vogais[5] = {‘a’, ‘e’, ‘i’, ‘o’, ‘u’};
```

```
float moedas[6] = {1.00, 0.50, 0.25, 0.10, 0.05, 0.01};
```

Se a quantidade de valores iniciais na declaração de um vetor for menor que seu tamanho, as demais posições são automaticamente zeradas.

Exercícios

1) Faça um algoritmo que preencha um vetor de números inteiros com 4 posições, imprima a soma desses valores.

2) Faça um algoritmo que leia uma sequencia de cinco números e imprima em ordem inversa.

3) Preencher um vetor com números inteiros(8 posições), solicitar um número do teclado, pesquisar se esse número existe no vetor. Se existir imprimir em qual posição do vetor ele está. Se não existir imprimir uma mensagem que não existe.

4) Faça um algoritmo que leia dois vetores (A e B) de seis elementos de números inteiros, faça a soma dos elementos dos vetores (A + B) e armazene o resultado em um terceiro vetor (C). O primeiro elemento do vetor A deve ser somado com o primeiro elemento do vetor B e assim sucessivamente.

Matrizes

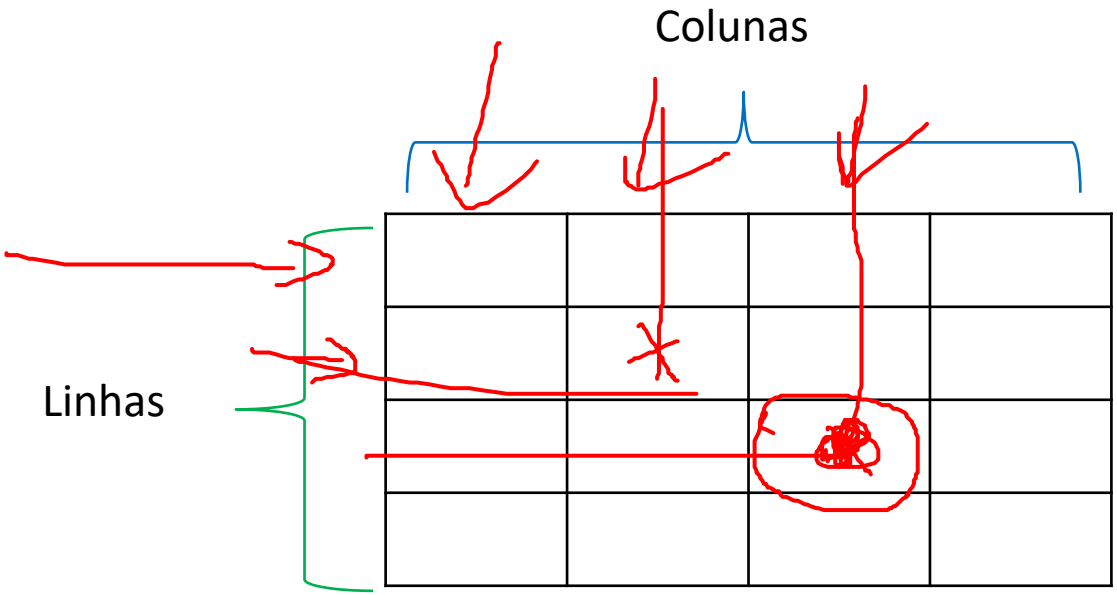
Uma matriz é um conjunto de vetores que pode ter qualquer número de dimensões. Quando essas estruturas tem duas dimensões são chamadas de **bidimensionais** e quando elas tem mais de duas dimensões são chamadas de **multidimensionais**.

Vejamos alguns exemplos:

- Tabuleiros de jogos em geral
- Matrizes matemáticas
- Todos os pontos definidos em um monitor (vídeo / TV)
- O volante de um jogo de loteria

Matrizes

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



Matrizes

Vejamos a declaração de uma Matriz:

<tipo_do_dado> <nome_da_matriz>[y][x];

Exemplo:

```
int matriz[ 4 ][ 2 ];
```

Matriz com valores constantes:

```
int matriz[ 4 ][ 2 ] = {{5,9},
                        {0,2}.
                        {6,-3}
                        {3,5}};
```

Matrizes

Preenchendo uma matriz 2 x 2

```
int matriz[ 2 ][ 2 ];
matriz[0][0] = 7;
matriz[0][1] = 8;
matriz[1][0] = 5;
matriz[1][1] = 9;
```

Matrizes

Para abordagens com estruturas pequenas podemos manipula-la acessando diretamente suas posições, como no exemplo anterior. Porém, essa abordagem se mostra inadequada quando manipularmos matrizes maiores como 100 x 100(10.000 elementos).

Para podermos resolver essa abordagem, precisamos manipular dois índices simultâneos: um para as linhas e outro para as colunas.

Matrizes

A estratégia utilizada é varrer todos os elementos de cada linha por vez. Ou seja, fixa a linha e leem-se todos os elementos dessa linha, variando o índice da coluna. Repete-se o processo para cada linha, até que todas as linhas sejam acessadas e, por consequência todos os elementos da matriz.

Matrizes

Portanto uma segunda opção seria criar um laço para tratar o índice das colunas. Vejamos o exemplo para uma matriz 2x2:

```
n_Col = 2;
For (int c = 0; c < n_Col; c++){
    matriz[1,c] = c;
}
For (int c = 0; c < n_Col; c++){
    matriz[2,c] = c;
}
```

Esta implementação é melhor que a primeira mas é necessário repetir varias vezes o mesmo procedimento.

Matrizes

A solução final seria criar um laço também para as linhas:

For (int l = 0; l < n_Lin; l++){

}

Matrizes

Se juntarmos tudo, teríamos:

n_Lin = 2;

n_Col = 2;

```
For (int l = 0; l < n_Lin; l++){
   For (int c = 0; c < n_Col; c++){
      matriz[l,c] = c;
   }
}
```

Exercícios

1) Faça um algoritmo que armazene o código e o nome de 5 funcionários de um empresa. Depois imprima-os.

2) Faça um algoritmo capaz de ler o RA e as notas de 3 testes de cada um dos 10 alunos de uma turma e indique o número de alunos cuja média é maior ou igual a 5.

Vetores - Ordenação

Faça um algoritmo que ordene os valores dentro de um vetor de tamanho ***n***.

Resolver este problema não é trivial. Porém, já existe uma serie de algoritmos conhecidos para resolve-lo. Vamos ver aqui dois algoritmos clássicos de ordenação:

Bolha (Bubblesort), um dos mais simples.

Quicksort, mais sofisticado e eficiente.

Bubblesort

A abordagem deste algoritmo consiste em varrer o vetor a partir do ultimo elemento trocando os elementos adjacentes que estejam fora de ordem. Ao final do processo o maior elemento necessariamente estará na ultima posição do vetor. Depois repete-se a operação até ordenar os demais elementos.

Bubblesort

```
for (int i = 4; i >= 1; i--){
    for (int j = 0; j < i; j++){
        if (vetA[j] < vetA[j+1]){
            aux = vetA[j];
            vetA[j] = vetA[j+1];
            vetA[j+1] = aux;
           }
    }
}
```

Quicksort

Esse algoritmo é baseado na ideia de “dividir para conquistar”. O vetor é dividido em duas partes que serão ordenados independentemente. Este processo repete-se até que o vetor inteiro esteja ordenado.

É o algoritmo de ordenação interna mais rápido que se conhece para uma ampla variedade de situações.

É muito interessante estuda-lo.

Vetores - Busca

A busca é um processo que determina se um item *x* está armazenado em um vetor v.

Existem alguns métodos de busca que podem ser utilizados para essa tarefa, dentre eles, temos a **busca linear** e a **busca binária**.

Busca linear

A **busca linear** é o método mais simples que existe. Ele não pressupõe nenhuma ordenação para o vetor e examina cada um dos itens do vetor até que o item desejado seja encontrado.

Busca binária

A **busca binária** é semelhante àquele método empregado quando procuramos uma palavra no dicionário. Primeiro examinamos uma pagina no meio do dicionário, se tivermos sorte e a palavra for encontrada, a busca termina, senão verificamos se a palavra procurada está antes ou depois da pagina que examinamos e continuamos, usando a mesma estratégia para procura-la na primeira ou na segunda parte do dicionário.

Vetores - Busca

Suponha que desejamos encontrar um item x em um vetor ordenado v, contendo n itens, e que v[m] seja um item aproximadamente ao meio de v. Então existem três possibilidades:

x == v[m]: neste caso o problema está resolvido

x < v[m]: neste caso devemos continuar procurando

na primeira metade de v

x > v[m]: neste caso devemos continuar procurando

na segunda metade de v.

Obrigado

NILTON.SACCO@FATEC.SP.GOV.BR