Estruturas de Dados I

Introdução

Prof. Bruno Azevedo

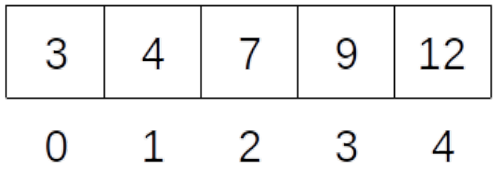
Instituto Federal de São Paulo

INSTITUTO FEDERAL DE
EDUCAÇÃO, CIÊNCIA E TECNOLOGIA
SÃO PAULO
Campus Catanduva

Vetores

- Um vetor (ou *array*) é uma coleção de elementos do mesmo tipo armazenados consecutivamente.
- A figura abaixo ilustra um vetor de 5 elementos, contendo os elementos 3, 4, 7, 9 e 12.

- Cada elemento do vetor é acessado por um índice inteiro que começa em zero.
- Ou seja, cada elemento pode ser acessado por seu *índice*.



Vetores

- Podemos declarar vetores de duas formas:
- Aqui, declaramos um vetor do tipo inteiro de 3 elementos.

```
int meuVetor[3];
```

- Aqui, declaramos um vetor do tipo inteiro de 3 elementos e atribuímos os valores 1, 2 e 3 para cada elemento do vetor.

```
int meuVetor[3] = {1, 2, 3};
```

- Também podemos declarar o vetor acima da seguinte forma:

```
int meuVetor[] = {1, 2, 3};
```

- As duas declarações acima são idênticas, e a primeira cria um vetor do mesmo tipo, e mesmo número de elementos, mas não atribui valores a seus elementos.
- Portanto, terá lixo nesses elementos.

Vetores

- Também podemos declarar um vetor inicializando todos os elementos com o mesmo valor.

```
int meuVetor[5] = {0};
```

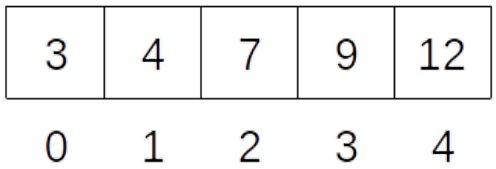
- Acima, declaramos um vetor de cinco elementos, todos serão inicializados com o valor zero.
- Entretanto, se escrevermos:

```
int meuVetor[] = {0};
```

- Aqui criamos um vetor de apenas uma posição, contendo o valor zero.
- Porque quando não declaramos o tamanho do vetor ele calcula de acordo com o número de elementos atribuídos.

Vetores

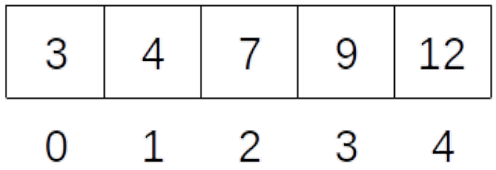
- Como dito, para acessarmos um elemento de um vetor, utilizamos seu índice, iniciando em zero.



- Por exemplo, podemos acessar o terceiro elemento do vetor através do índice 2.
- A sintaxe para acessar elementos do vetor é a seguinte:

  nomeVetor[índice].
- Ou seja, usa-se o nome do vetor seguido do índice entre colchetes.
- Com isso, podemos ler ou modificar elementos do nosso vetor.

Vetores



- Por exemplo, suponha que a figura acima representa um vetor de nome "meuVetor":

```
x = meuVetor[3];
meuVetor[1] = 5;
meuVetor[0] = meuVetor[3] + meuVetor[4];
```

- No trecho de código acima:
  - A variável x recebe o valor do quarto elemento do vetor.
  - O segundo elemento do vetor recebe o valor cinco.
  - O primeiro elemento do vetor (ou elemento de índice zero) recebe o valor 21.

Vetores

- Podemos usar laços para ler, ou modificar, os elementos de um vetor.
- Qual será o resultado da execução deste programa? Vamos pensar juntos.

```
#include <iostream>
using namespace std;
int main() {
    int meuVetor[5] = {3, 4, 7, 9, 12};
    for (int i = 0; i < 5; i++)
        if (i < 4)
            meuVetor[i] += meuVetor[i + 1];
        else
            meuVetor[i] += meuVetor[i];
    for (int i = 0; i < 5; i++)
        cout << "meuVetor["<< i << "]: " << meuVetor[i] << endl;
    return 0;
}
```

Vetores

- Podemos usar laços para ler, ou modificar, os elementos de um vetor.
- Qual será o resultado da execução deste programa? Vamos pensar juntos.

```
#include <iostream>
using namespace std;
int main() {
    int meuVetor[5] = {3, 4, 7, 9, 12};
    for (int i = 0; i < 5; i++)
        if (i < 4)
            meuVetor[i] += meuVetor[i + 1];
        else
            meuVetor[i] += meuVetor[i];
    for (int i = 0; i < 5; i++)
        cout << "meuVetor["<< i << "]: " << meuVetor[i] << endl;
    return 0;
}
```

```
meuVetor[0]: 7
meuVetor[1]: 11
meuVetor[2]: 16
meuVetor[3]: 21
meuVetor[4]: 24
```

- E vocês sabem me dizer por que eu fiz um cálculo diferente para o último elemento do vetor?

Vetores

- Para calcularmos o tamanho de um vetor podemos usar o operador **sizeof**.
- Entretanto, ele nos retornará o tamanho alocado para o vetor e não quantos elementos existem no vetor.
- Portanto, devemos dividir este valor pelo tamanho de cada elemento.

```
#include <iostream>
using namespace std;
int main() {
    int vetor[] = {16,64,256,1024,4096};
    cout << sizeof(vetor) << endl; //No godbolt.org, 20
    cout << sizeof(vetor)/sizeof(int) << endl; // 5
}
```

- Como temos um vetor de inteiros, precisamos dividir pelo tamanho de um inteiro.
- No exemplo acima, temos que o compilador e a arquitetura utilizam 4 bytes por int, portanto, o espaço alocado é de 20 bytes para o vetor de 5 posições.

Vetores Multidimensionais

- Da mesma forma que declaramos vetores, podemos declarar vetores multidimensionais.
- Imagino que vocês conheçam matrizes, por exemplo.
- Uma matriz é uma estrutura de dados que organiza elementos do mesmo tipo em uma grade bidimensional, com linhas e colunas.
- Ou seja, é um vetor multidimensional de duas dimensões.
- Na figura abaixo temos uma matriz de 3 linhas por 4 colunas.

3	4	7	9
2	5	1	0
6	3	8	5

Vetores Multidimensionais

- A declaração de vetores multidimensionais é análoga à de vetores unidimensionais:
- Por exemplo, aqui declaramos uma matriz de inteiros com 3 linhas e 4 colunas.

```
int minhaMatriz[3][4];
```

- Como esta é uma matriz 3×4, ela contém 12 elementos.
- Como em vetores unidimensionais, efetuamos a atribuição de valores usando os índices.
- Neste caso, precisamos considerar os índices de cada dimensão para fazer a atribuição.

Vetores Multidimensionais

- Vamos atribuir valores para todos os elementos de nossa matriz.

```
minhaMatriz[0][0] = 3;
minhaMatriz[0][1] = 4;
minhaMatriz[0][2] = 7;
minhaMatriz[0][3] = 9;

minhaMatriz[1][0] = 2;
minhaMatriz[1][1] = 5;
minhaMatriz[1][2] = 1;
minhaMatriz[1][3] = 0;

minhaMatriz[2][0] = 6;
minhaMatriz[2][1] = 3;
minhaMatriz[2][2] = 8;
minhaMatriz[2][3] = 5;
```

3	4	7	9
2	5	1	0
6	3	8	5

Vetores Multidimensionais

- Para inicializarmos durante a criação da variável, podemos organizar com chaves.

```
int minhaMatriz[3][4] = {
      {3, 4, 7, 9},
      {2, 5, 1, 0},
      {6, 3, 8, 5}
  };
```

- Ou podemos simplesmente listar todos os valores em sequência.

```
int minhaMatriz[3][4] = {3, 4, 7, 9, 2, 5, 1, 0, 6, 3, 8, 5};
```

Vetores Multidimensionais

- Não estamos restritos a duas dimensões. O código abaixo cria um vetor multidimensional de quatro dimensões ($2 \times 2 \times 2 \times 2$).

```
#include <iostream>
using namespace std;
int main() {
 const int d1 = 2, d2 = 2, d3 = 2, d4 = 2;/* Declarando variáveis constantes
                                              do tipo int. */
 int vetor4D[d1][d2][d3][d4];
 for (int i = 0; i < d1; ++i) {
  for (int j = 0; j < d2; ++j) {
   for (int k = 0; k < d3; ++k) {
    for (int l = 0; l < d4; ++l) {
     // O que será atribuído a cada elemento?
     vetor4D[i][j][k][l] = i * d2 * d3 * d4 + j * d3 * d4 + k * d4 + l + 1;
     cout << "vetor4D[" << i << "][" << j << "][" << k << "][" << l << "]: "
          << vetor4D[i][j][k][l] << endl;
    }
   }
  }
 }
 return 0;
} // Tentem descobrir o que é atribuído a cada posição do vetor sem executar este pr
```

Vetores

Exercícios (8).

⇒ Vetores

- Escreva um programa que solicite 10 valores inteiros do usuário e armazene-os em um vetor.
- Escreva um programa que utilize o vetor anterior e calcule a soma de todos os seus elementos.
- Escreva um programa que utilize o vetor anterior e exiba o maior e o menor valor.
- Escreva um programa que inicie com um vetor com 10 valores inteiros e exiba a média dos valores ímpares. Verifique se há valores ímpares no vetor.

Vetores

Exercícios (9).

⇒ Vetores

- Sem executar o código abaixo em um compilador, responda: o que será exibido ao fim de sua execução?

```
#include <iostream>
using namespace std;
int main() {
    int meuVetor[] = {21, 7, -21, -9, 76, -17, 41};
    int s1 = meuVetor[5];
    int s2 = meuVetor[3];
    while (s1 < s2) {
        s1 += 3;
        s2 -= 2;
    }
    meuVetor[1] = s1 + s2;
    meuVetor[4]  = s1;
    meuVetor[5]  = s2;
    for (int i = 0; i < 7; i++)
        cout << "meuVetor["<< i << "]: " << meuVetor[i] << endl;
    return 0;
}
```