

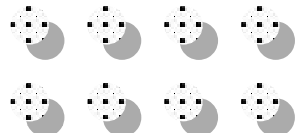
OBI

Aula 04

GEMP
Grupo de Estudos da
Maratona de Programação

Tópicos

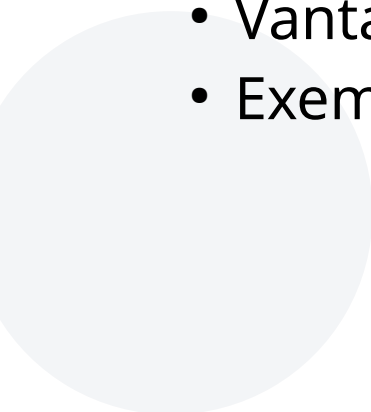
- O que vimos?
- Matrizes
- Vectors
- Matrizes **Vs** Vectors
- Exemplo
- Exercícios





O que vimos...



- **Vetores** (ou Arrays)
 - Como declarar
 - Vantagens
 - Exemplos
- 



```
#include <iostream>
using namespace std;

int main() {

    int a[] = {2,3};

    for (int i = 0; i < 2; i++)
        cout << a[i] << " ";
    cout << endl;

    return 0;
}
```



Codelmage

```
#include <iostream>
using namespace std;

int main() {

    int b[2];

    for (int i = 0; i < 2; i++)
        cout << b[i] << " ";
    cout << endl;

    return 0;
}
```



CodelImage

```
#include <iostream>
using namespace std;

int main() {

    int c[2] = {2}; // [2, 0]

    for (int i : c)
        cout << i << " ";
    cout << endl;

    return 0;
}
```

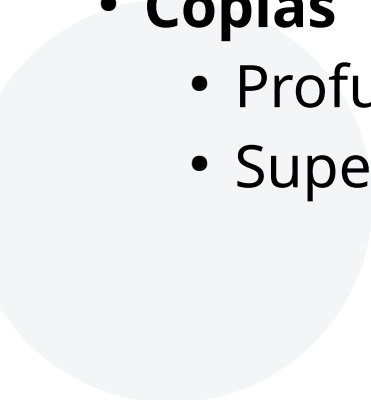


CodelImage



O que vimos...



- **Ponteiros e Referências**
 - **Copias**
 - Profundas (deep copy)
 - Superficiais (shallow copy)
- 



```
#include <iostream>
using namespace std;

int main() {

    int d = 3;
    cout << d << endl; // 3

    int* e = &d;
    cout << *e << endl; // 3

    *e = 2;
    cout << *e + d << endl; // 4 ← 2 + 2

    return 0;
}
```



Codelmage


```
#include <iostream>
using namespace std;

int main() {

    int f[3];           // [0, 0, 0]
    for (int i : f)
        cout << i << " ";
    cout << endl;

    int* g = f;
    for (int i = 0; i < 3; i++)
        cout << g[i] << " ";
    cout << endl;


    g[0] = 1;
    f[1] = 10;
    for (int i : f)     // [1, 10, 0]
        cout << f[i] << " ";
    cout << endl;

    return 0;
}
```

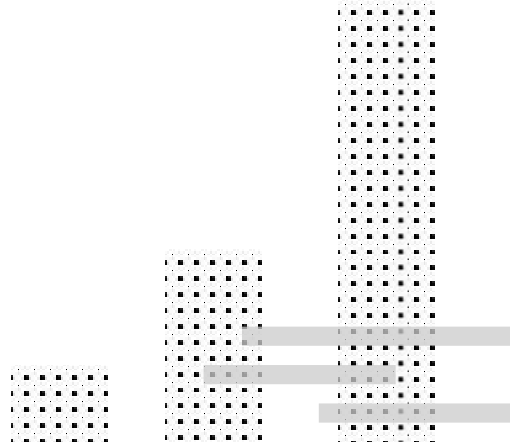
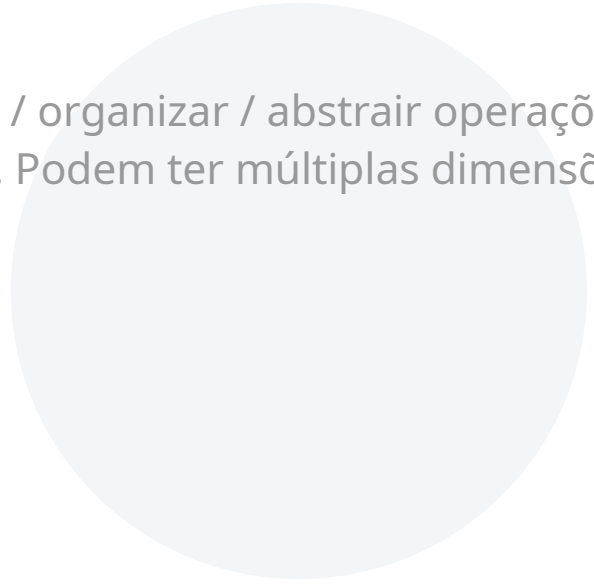


Matrizes

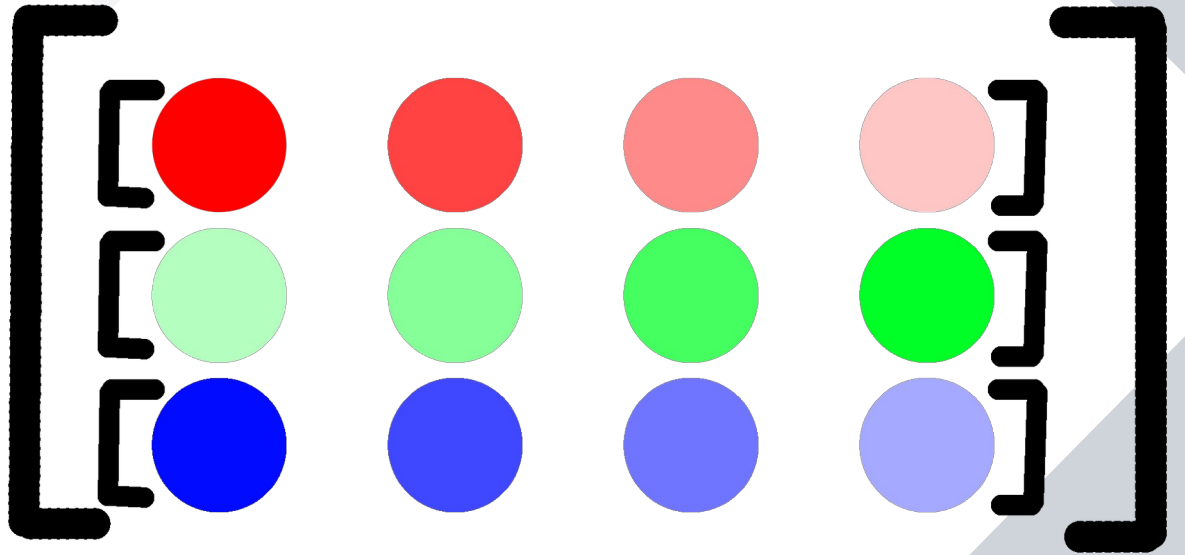
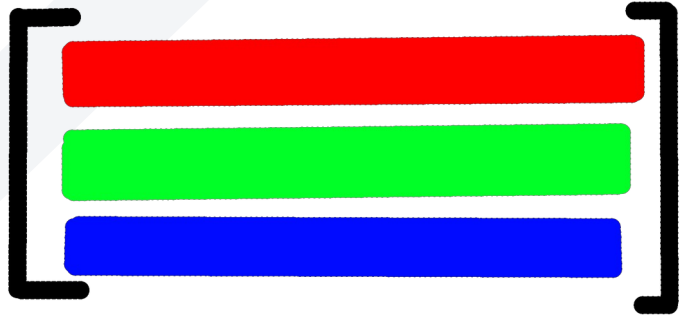
Matrizes



Servem para simplificar / organizar / abstrair operações com os vetores (arrays). Podem ter múltiplas dimensões.



Matriz na Matemática



Matriz na Computação

$[[\text{red circle}, \text{red circle}, \text{light red circle}, \text{light red circle}], [\text{light green circle}, \text{light green circle}, \text{green circle}, \text{green circle}], [\text{blue circle}, \text{blue circle}, \text{light blue circle}, \text{light blue circle}]]$



Vetor / Array:

[1, 2, 10, 20]

==

Matriz 2x2:

[[1, 2], [10, 20]]





Vetor / Array:

[1, 2, 3, 1, 8, 27]

==

Matriz 2x3:

[[1, 2, 3], [1, 8, 27]]





Vetor / Array:

[1, 2, 3, 4, 2, 4, 6, 8]

==

Matriz 2x4:

[[1, 2, 3, 4], [2, 4, 6, 8]]





Vetor / Array:

[1, 2, 3, 4, 2, 4, 6, 8]

==

Matriz 4x2:

[[1, 2], [3, 4], [2, 4], [6, 8]]




```
#include <iostream>
using namespace std;

int main() {

    int a[2][2] = {{1,2},{10,20}};
    int b[][2] = {{1,2},{10,20}};

    cout << (a == b) << endl;           // 0 → False

    int (*ponteiro_de_a)[2] = a;

    cout << (ponteiro_de_a == a) << endl; // 1 → True
    return 0;
}
```

Exemplo

- Fazer um programa capaz de armazenar os nomes dos alunos presentes nos encontros do POBI, por meio de listas de nomes (por mês e encontro).
- Também fazer com que receba as presenças do primeiro encontro (podendo parar de receber nomes quando receber a palavra “fim”) e mostre os nomes ao final da execução.

```

#include <iostream>
using namespace std;

int main() {
    int inicio_da_pobi = 4;
    int fim_do_semestre = 7;
    int meses_de_pobi = fim_do_semestre - inicio_da_pobi;

    int max_de_segundas_por_mes = 5;
    int max_numero_de_alunos = 20;

    string presenca_encontro_por_mes[meses_de_pobi][max_de_segundas_por_mes][max_numero_de_alunos];

    // string (*presenca_primeiro_encontro) = presenca_primeiro_mes[0][0];
    string (*presenca_primeiro_mes)[max_numero_de_alunos] = presenca_encontro_por_mes[0];
    string (*presenca_primeiro_encontro) = presenca_primeiro_mes[0];

    for (int i = 0; i < max_numero_de_alunos; i++) {
        string nome;
        getline(cin, nome);

        if (nome == "fim")
            break;

        presenca_primeiro_encontro[i] = nome;
    }

    for (int i = 0; i < max_numero_de_alunos; i++) {
        string nome = presenca_primeiro_encontro[i];
        if (nome != "")
            cout << nome << endl;
    }

    return 0;
}

```

Exercício

Fazer um programa que receba um número N (inteiro ≥ 2) que representa a quantidade de números inteiros que o programa deve receber para que formem matrizes de dimensões $m \times 2$ (se N for par) ou $m \times 3$ (se N for ímpar) – sendo m um número natural ($m > 0$).

– Ex.:

Input $\rightarrow 4 \ 1 \ 2 \ 3 \ 4$

Teste: $\text{matriz}[0][1] \rightarrow 2$ e $\text{matriz}[1][0] \rightarrow 3$


Input $\rightarrow 5 \ 99 \ 20 \ 90 \ 11 \ 42$

Teste $\rightarrow \text{matriz}[0][2] \rightarrow 90$ e $\text{matriz}[1][2] \rightarrow 0$

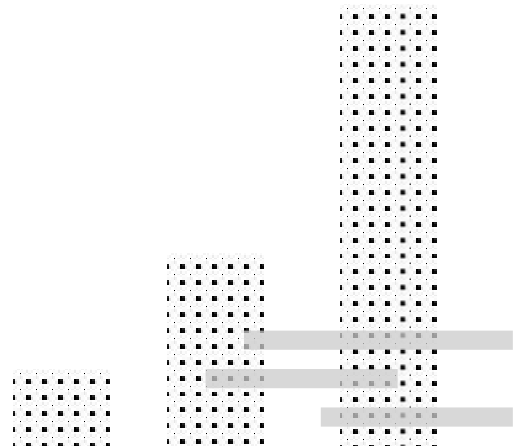
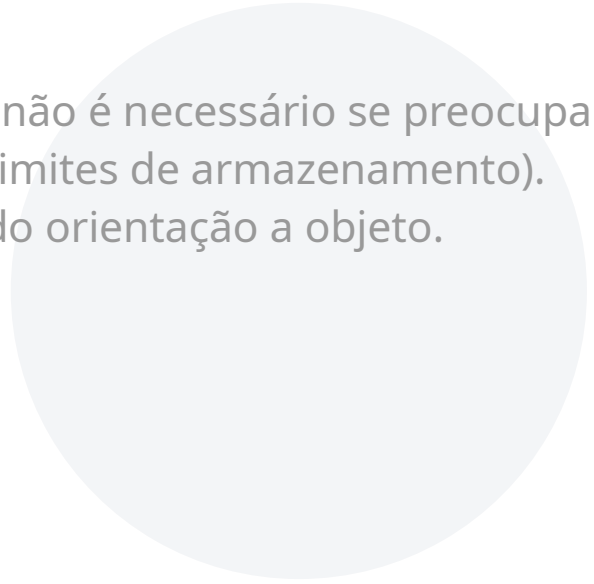


Vectors

Vectors



São vetores dinâmicos (não é necessário se preocupar diretamente com seus limites de armazenamento).
Implementado utilizando orientação a objeto.



.size()

Retorna o número de elementos armazenados no Vector.

.at()

Retorna o valor do elemento armazenado na posição especificada.

.pop_back()

Remove o último valor do Vector, sem retorno.

.push_back()

Adiciona um novo valor ao final do Vector.

.front()

Retorna o primeiro valor do Vector.

.back()

Retorna o último valor do Vector.



```
#include <iostream>
using namespace std;

int main() {

    vector<int> vec;

    for (int i : vec)
        cout << i << " "; // Nada
    cout << endl;

    return 0;
}
```



Codelmage

```
#include <iostream>
using namespace std;

int main() {

    vector<int> vec(3); // [0, 0, 0]

    for (int i : vec)
        cout << i << " ";
    cout << endl;

    return 0;
}
```



Codelmage


```
#include <iostream>
using namespace std;

int main() {

    vector<int> vec(3, {2}); // [2, 2, 2]

    for (int i : vec)
        cout << i << " ";
    cout << endl;

    return 0;
}
```



CodelImage

```
#include <iostream>
using namespace std;

int main() {

    vector<int> vec = {2, 3, 4}; // [2, 3, 4]

    for (int i : vec)
        cout << i << " ";
    cout << endl;

    return 0;
}
```



CodelImage

```
#include <iostream>
using namespace std;

int main() {

    vector<int> vec({2, 3, 4});

    vector<int> vec2;    // []
    vec2.push_back(2);   // [2]
    vec2.push_back(3);   // [2, 3]
    vec2.push_back(4);   // [2, 3, 4]

    cout << endl;
    cout << (vec2 == vec) << endl;

    return 0;
}
```



CodelImage

```
#include <iostream>
using namespace std;

int main() {

    vector<int> vec{2, 3, 4, 5}; // [2, 3, 4, 5]

    for (int i = 0; i < vec.size(); i++) {

        cout << vec.at(i) << " ";

        if (i % 2 != 0)
            vec.at(i) -= 1; // vec.at(i) = vec.at(i) - 1
    }
    cout << endl;

    for (int i : vec) // [2, 2, 4, 4]
        cout << i << " ";
    cout << endl;

    return 0;
}
```

```
#include <iostream>
using namespace std;

void print_vector(vector<int> &vec);

int main() {

    vector<int> vec{1, 2, 3}; // [1, 2, 3]

    vector<int> copia_profunda = vec;

    vec.pop_back();

    print_vector(vec);           // [1, 2]
    print_vector(copia_profunda); // [1, 2, 3]

    return 0;
}

void print_vector(vector<int> &vec) {
    for (int i : vec)
        cout << i << " ";
    cout << endl;
}
```

```
#include <iostream>
using namespace std;

void print_vector(vector<int> vec);

int main() {

    vector<int> vec{1, 2, 3}; // [1, 2, 3]

    vector<int>* copia_superficial = &vec;

    vec.pop_back();           // [1, 2, 3] - [3]
    copia_superficial->pop_back(); // [1, 2] - [2]

    print_vector(vec);        // [1]
    print_vector(*copia_superficial); // [1]

    return 0;
}

void print_vector(vector<int> &vec) {
    for (int i : vec)
        cout << i << " ";
    cout << endl;
}
```

```
#include <iostream>
using namespace std;

void print_vector(vector<int> &vec);

int main() {

    vector<int> vec{1, 2, 3}; // [1, 2, 3]

    print_vector(vec); // [1, 2, 3]
    print_vector(vec); // [1, 2]

    return 0;
}

void print_vector(vector<int> &vec) {
    for (int i : vec)
        cout << i << " ";
    cout << endl;

    vec.pop_back();
}
```

```
#include <iostream>
using namespace std;

void print_vector(vector<int> vec);

int main() {

    vector<int> vec{1, 2, 3}; // [1, 2, 3]

    print_vector(vec); // [1, 2, 3]
    print_vector(vec); // [1, 2, 3]

    return 0;
}

void print_vector(vector<int> vec) {
    for (int i : vec)
        cout << i << " ";
    cout << endl;

    vec.pop_back();
}
```



```
#include <iostream>
using namespace std;

int main() {

    vector<vector<int>> vec{{1, 2}, {30, 40}}; // [[ 1,  2],
                                              // [30, 40]]

    for (vector<int> v : vec) {
        for (int i : v)
            cout << i << " ";
        cout << endl;
    }

    return 0;
}
```

```
#include <iostream>
using namespace std;

int main() {

    vector<vector<int>> vec{{1, 2}, {30, 40}}; // [[ 1,  2],
                                              // [30, 40]]

    vec.pop_back(); // [[ 1,  2], - [[30, 40]] = [[1, 2]]
                  // [30, 40]]

    vec.push_back({0,0}); // [[ 1,  2]] + [[0, 0]] = [[1, 2]]
                        //                               [0, 0]]

    for (auto v : vec) {
        for (int i : v)
            cout << i << " ";
        cout << endl;
    }

    return 0;
}
```

```
#include <iostream>
using namespace std;

int main() {

    vector<vector<int>> vec{{1, 2}, {30, 40}}; // [[ 1,  2],
                                              // [30, 40]]

    auto copia_profunda = vec;

    copia_profunda.at(0) = {0, 0};           // [[ 1,  2],  =>  [[ 0,  0],
                                              // [30, 40]]      // [30, 40]]

    for (auto v : copia_profunda) {
        for (int i : v)
            cout << i << " ";
        cout << endl;
    }

    return 0;
}
```



CodelImage

```
#include <iostream>
using namespace std;

int main() {

    vector<vector<int>> vec{{1, 2}, {30, 40}}; // [[ 1, 2],
                                              // [30, 40]]

    auto* copia_superficial = &vec;

    for (auto v : *copia_superficial) {
        for (int i : v)
            cout << i << " ";
        cout << endl;
    }

    return 0;
}
```

```
#include <iostream>
using namespace std;

int main() {

    vector<vector<int>> vec{{1, 2}, {30, 40}}; // [[ 1, 2],
                                              // [30, 40]]

    auto* copia_superficial = &vec;

    //vec.erase(vec.begin());
    copia_superficial->erase(copia_superficial->begin()); // [[ 1, 2], - [[1, 2]] = [[30, 40]]
                                                         // [30, 40]]

    for (auto v : *copia_superficial) {
        for (int i : v)
            cout << i << " ";
        cout << endl;
    }

    return 0;
}
```

Exercício

Fazer um programa que receba N números naturais ($N > 0$) que irão compor matrizes de dimensões $m \times 2$ (se N for par) ou $m \times 3$ (se N for ímpar) – sendo m também um número natural. O programa deve parar de receber até que ele receba um valor que não seja um número natural.

– Ex.:

Input $\rightarrow 1\ 2\ 3\ 4\ -1$

Teste: `vec.at(0).at(1)` $\rightarrow 2$ e `vec.at(1).at(0)` $\rightarrow 3$

Input $\rightarrow 99\ 20\ 90\ 11\ 42\ -100$

Teste \rightarrow `vec.at(0).at(2)` $\rightarrow 90$ e `vec.at(1).at(2)` $\rightarrow 0$