



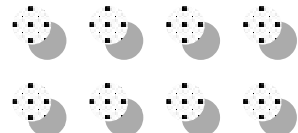
# **POBI**

## Aula 03 - Arrays

Preparatório da Olimpíada  
Brasileira de Informática  
15 de abril de 2024

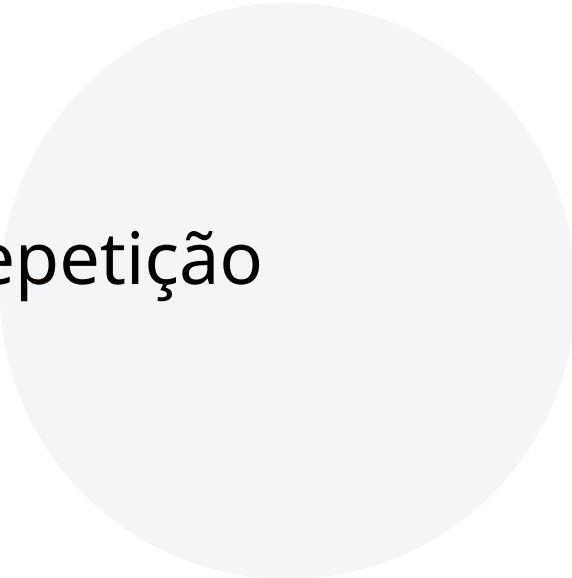
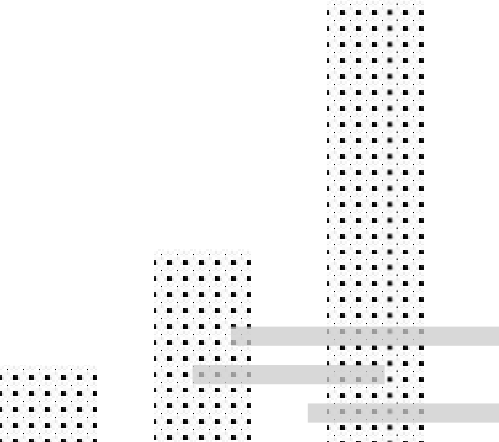
# Sumário

- O que vimos?
- Por que usar *array*?
- Definição
- Implementação
- Exemplo de Uso
- Vantagens
- Exercícios





# O que vimos?

- Estruturas de Decisão
    - If, Else If e Else
    - Switch
  - Estruturas de Repetição
    - For
    - While
- 
- 

# Por que usar *array*?

Suponha que você é um professor e precisa registrar as notas de 20 alunos em uma disciplina.

Sem o uso de *arrays*, você poderia declarar 20 variáveis separadas para cada aluno, como "**nota\_aluno1**", "**nota\_aluno2**", "**nota\_aluno3**", e assim por diante. No entanto, isso tornaria seu código extenso, repetitivo e difícil de manter.

```
1 float nota_aluno1 = 7.0;
2 float nota_aluno2 = 8.5;
3 float nota_aluno3 = 3.4;
4 float nota_aluno4 = 6.0;
5 float nota_aluno5 = 0.4;
6 // ...
```


# Por que usar *array*?

Por outro lado, ao utilizar um *array*, você pode armazenar todas as notas dos alunos em uma única estrutura de dados. Por exemplo, em C++:

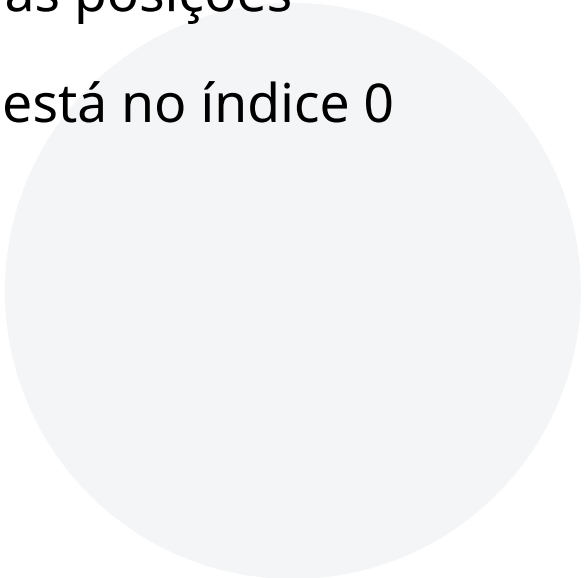
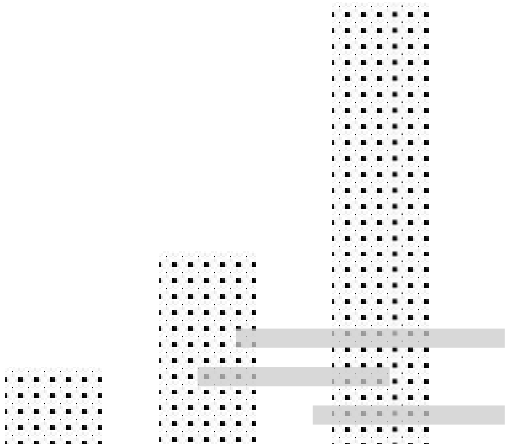
```
float notas_alunos[20];
```

Neste caso, o array "notas\_alunos" pode armazenar as notas de todos os 20 alunos, e você pode acessar essas notas facilmente usando índices.

```
1 // Declarando um array do tipo float
2 // chamado notas_alunos,
3 // de tamanho 20.
4 float notas_alunos[20];
5
6 // Atribuindo notas aos alunos
7 notas_alunos[0] = 8.5; // aluno 1
8 notas_alunos[1] = 7.0; // aluno 2
9 // ...
10
11 // Acessando a nota de um aluno específico
12 float nota_aluno2 = notas_alunos[1];
```

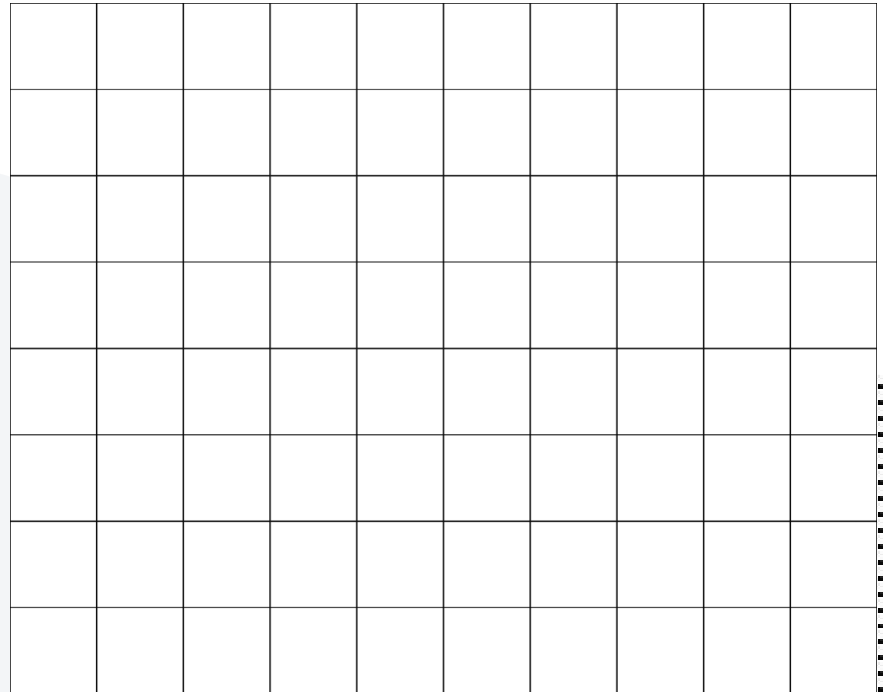


# ***Arrays ou Vetores***

- Estruturas de dados unidimensionais
  - Índice único controla as posições
  - O primeiro elemento está no índice 0
- 
- 

# ***Arrays ou Vetores***

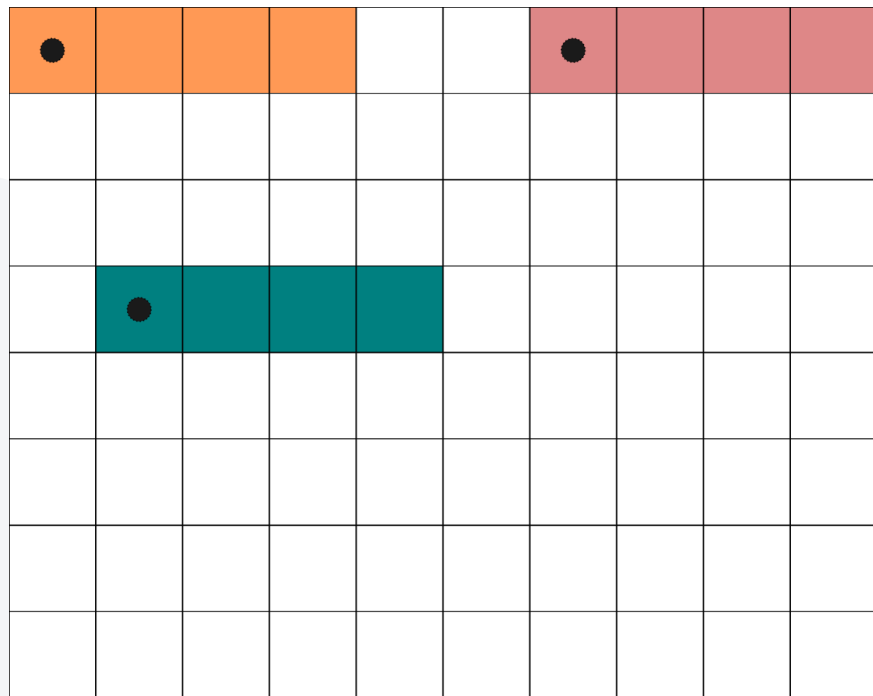
- A memória do computador é organizada da seguinte forma:
- Cada bloco represente um byte (8 bits)



# Arrays ou Vetores

- Ao declararmos variáveis do tipo int, ocupamos 4 espaços contínuos da memória.
- Note que não necessariamente elas estão uma após a outra na memória
- Neste exemplo:

```
int nota1, nota2, nota3;
```





# Arrays ou Vetores

- Ao declararmos um array, é alocado na memória  $n * t$  ( $t$  = o tamanho necessário para o tipo do vetor) espaços contínuos, que serão acessados mediante ao ponteiro + índice.
  - Como é um vetor de inteiros, são alocados  $n * 4$  bytes na memória.
  - No exemplo ao lado, temos:
- ```
int notas[5];
```

[illegible]

# Implementação

- Sintaxe de declaração: `<tipo> <nome>[<tamanho>];`
- Exemplo:

```
float notas_alunos[20];
```

```
int fibonacci[10];
```

# Implementação

- Para acessar uma posição do vetor:

```
<tipo> <nome>[<índice>;
```

```
float notas[2];
```

- Podemos inicializar usando uma lista:

```
float notas[3] = {9, 3.7, 5};
```

- Note que a quantidade de valores da lista tem que ser necessariamente o tamanho do vetor.

# Exemplo de Uso



```
1  #include <iostream>
2  using namespace std;
3
4  int main() {
5      string alunos[4];
6      alunos[0] = "João";
7      alunos[1] = "Maria";
8      alunos[2] = "José";
9      alunos[3] = "Ana";
10
11     for (int i = 0; i < 4; i++) {
12         cout << "Aluno " << i + 1 << ": " << alunos[i] << endl;
13     }
14 }
```

Saída:

Aluno 1: João

Aluno 2: Maria

Aluno 3: José

Aluno 4: Ana

# Vantagens do uso de arrays

## Organização de Dados

Os arrays permitem organizar dados de forma estruturada e sequencial, facilitando o acesso e a manipulação.

## Acesso Rápido

O acesso aos elementos de um array é feito por meio de índices, o que proporciona acesso direto e rápido a qualquer elemento, independentemente da sua posição no array.

## Eficiência

Ao utilizar um único array para armazenar múltiplos elementos de dados do mesmo tipo, economizamos recursos de memória e processamento.

## Iteração Simples

Os arrays facilitam a iteração sobre conjuntos de dados, permitindo que operações sejam realizadas em todos os elementos de forma eficiente.



# Exercícios

- Faça um programa que **leia** um vetor  $X[10]$ . Substitua a seguir, todos os valores nulos e negativos do vetor  $X$  por 1. Em seguida mostre o vetor  $X$ .
- **Entrada:** A entrada contém 10 valores inteiros, podendo ser positivos ou negativos.
- **Saída:** Para cada posição do vetor, escreva " $X[i] = x$ ", onde  $i$  é a posição do vetor e  $x$  é o valor armazenado naquela posição.

## Exemplo de Entrada

```
0
-5
63
0
...
```

## Exemplo de Saída

```
X[0] = 1
X[1] = 1
X[2] = 63
X[3] = 1
...
```

# Exercícios

- Leia um valor e faça um programa que coloque o valor lido na primeira posição de um vetor  $N[10]$ . Em cada posição subsequente, coloque o dobro do valor da posição anterior. Mostre o vetor em seguida.
- **Entrada:** A entrada contém um valor inteiro ( $V \leq 50$ ).
- **Saída:** Para cada posição do vetor, escreva " $N[i] = X$ ", onde  $i$  é a posição do vetor e  $X$  é o valor armazenado na posição  $i$ . O primeiro número do vetor  $N$  ( $N[0]$ ) irá receber o valor de  $V$ .

## Exemplo de Entrada

1

## Exemplo de Saída

$N[0] = 1$

$N[1] = 2$

$N[2] = 4$

...