

# Fundamentos de Programação

Matrizes



JESUÍTAS BRASIL



Somos infinitas possibilidades

	Aluno		Nota1		Nota2		Nota3		Nota4		Média
0	Pedro	0	5.6	0	6.0	0	7.3	0	5.6	0	6.1
1	Ana	1	10	1	4.0	1	5.0	1	7.3	1	6.6
2	Luiz	2	4.5	2	2.0	2	5.5	2	1.0	2	3.3
...	...	...	...	...	...	...	...	...	...	...	...
48	Matheus	48	7.2	48	6.6	48	8.1	48	8.8	48	7.7
49	Andre	49	6.0	49	9.0	49	7.3	49	4.5	49	6.6



JESUÍTAS BRASIL

**UNISINOS**

Somos infinitas possibilidades

E se tivermos que armazenar 100 notas?

	Aluno		Nota1		Nota2		Nota3				Nota99		Nota100		Média	
0	Pedro	0	5.6	0	6.0	0	7.3				0	7.3	0	5.6	0	6.1
1	Ana	1	10	1	4.0	1	5.0				1	5.0	1	7.3	1	6.6
2	Luiz	2	4.5	2	2.0	2	5.5				2	5.5	2	1.0	2	3.3
...	...	...	...	...	...	...	...		...		...	...	...	...	...	...
48	Matheus	48	7.2	48	6.6	48	8.1				48	8.1	48	8.8	48	7.7
49	Andre	49	6.0	49	9.0	49	7.3				49	7.3	49	4.5	49	6.6

Criaremos 100 vetores com 100 nomes diferentes?



JESUÍTAS BRASIL

UNISINOS

Somos infinitas possibilidades

Uma solução mais eficaz para resolver o problema é o uso de **matrizes**:

Aluno		0	1	2	...	98	99	Média		
0	Pedro	0	5.6	6.0	7.3	...	7.3	5.6	0	6.1
1	Ana	1	10	4.0	5.0	...	5.0	7.3	1	6.6
2	Luiz	2	4.5	2.0	5.5	...	5.5	1.0	2	3.3
...	...	...	...	...	...	...	...	...	...	...
48	Matheus	48	7.2	6.6	8.1	...	8.1	8.8	48	7.7
49	Andre	49	6.0	9.0	7.3	...	7.3	4.5	49	6.6



`<tipo> <identificador> [<linhas>] [<colunas>];`

- `<tipo>`: tipo dos dados que serão armazenados no vetor (int, char, float, etc);
- `<identificador>`: nome dado à variável;
- `<linhas>`: número de elementos da primeira dimensão;
- `<colunas>`: número de elementos da segunda dimensão;
- As linhas e colunas são numeradas de 0 até *tamanho* – 1.



Uma matriz é uma estrutura de dados de **variáveis de mesmo tipo**, acessíveis com um **único nome** e armazenados contiguamente na memória.

A individualização de cada variável de um vetor é feita através do uso de **índices**.

Os **Vetores** são matrizes de uma só dimensão.

```
int Vetor[5];    // declara um vetor de 5 posições  
  
int Matriz[5][3]; // declara uma matriz de 5 linhas e 3 colunas
```



JESUÍTAS BRASIL



Somos infinitas possibilidades

Uma matriz é uma estrutura de dados de **variáveis de mesmo tipo**, acessíveis com um **único nome** e armazenados contiguamente na memória.

A individualização de cada variável de um vetor é feita através do uso de **índices**.

```
int Vetor[5]; // declara um vetor de 5 posições

int Matriz[5][3]; // declara uma matriz de 5 linhas e 3 colunas

Vetor[0] = 9; // coloca 9 na primeira posição do vetor
Vetor[4] = 30 // coloca 30 na última posição do vetor

Matriz[0][1] = 15; // coloca 15 na célula que está na primeira linha
                  // e na segunda coluna da matriz
```



JESUÍTAS BRASIL



Somos infinitas possibilidades

Podemos também definir constantes para o tamanho da matriz.

```
#define NLIN 10  
#define NCOL 10  
  
int Matriz[NLIN][NCOL];
```





## Preenchendo uma matriz

```
for(i=0; i < NLIN; i++){  
    for(j=0; j < NCOL; j++){  
        Matriz[i][j] = 30;  
    }  
}
```



```
//imprimir o elemento da linha 3 e coluna 10 da matriz notas  
printf("%lf", notas[3][10]);  
  
//multiplica a posição (i, j) da matriz mat por 5;  
mat[i][j] = mat[i][j] * 5;
```



JESUÍTAS BRASIL



Somos infinitas possibilidades

```
// Declaração de uma matriz 3x4, inicializada com zeros
int matriz[3][4] = {0};

// Inicialização explícita da matriz com valores específicos
int matrizExemplo[3][4] = {
    {1, 2, 3, 4}, // Primeira linha
    {5, 6, 7, 8}, // Segunda linha
    {9, 10, 11, 12} // Terceira linha
};
```



```
// Imprimindo os valores da matrizExemplo
for(int i = 0; i < 3; i++) {
    for(int j = 0; j < 4; j++) {
        printf("%d ", matrizExemplo[i][j]);
    }
    printf("\n"); // Quebra de linha para cada nova linha da matriz
}
```



JESUÍTAS BRASIL



Somos infinitas possibilidades

## Lendo como input do teclado

```
int matriz[3][3];

printf("Digite os valores para uma matriz 3x3:\n");

// Loop para ler os valores da matriz do teclado
for(int i = 0; i < 3; i++) {
    for(int j = 0; j < 3; j++) {
        printf("Elemento [%d][%d]: ", i, j);
        scanf("%d", &matriz[i][j]);
    }
}
```



Exercício 1 – Faça um programa que leia os elementos de uma matriz dados pelo usuário e imprima ela na tela



JESUÍTAS BRASIL



Somos infinitas possibilidades

Dada uma matriz ( $4 \times 5$ ), calcular a soma de todos os elementos da matriz. Calcular também o somatório dos elementos de cada linha da matriz, armazenando o somatório em um vetor.

### SOMALINHA

**MAT**

1	2	3	4	5
0	-1	0	-3	1
2	-2	-2	2	0
0	0	6	0	0

15
-3
0
6



JESUÍTAS BRASIL



Somos infinitas possibilidades

# Inicialização de matrizes pt 2

Inicializando na declaração. Processo semelhante à inicialização de vetores.

```
1  int matriz[3][4] = { {10, 20, 30, 40},  
2                        {50, 60, 70, 80},  
3                        {90, 11, 22, 33} };
```

Mas podemos fazer também:

```
1  int matriz[3][4] = { 10, 20, 30, 40,  
2                        50, 60, 70, 80,  
3                        90, 11, 22, 33 };
```

Ou ainda:

```
1  int matriz[3][4] = { 10, 20, 30, 40, 50, 60, 70, 80, 90, 11, 22, 33 };
```





# Funções e Matrizes

```
void imprimirMatriz(int matriz[3][3], int n, int m)
{
    for (int i = 0; i < n; ++i) {
        for (int j = 0; j < m; ++j){
            printf("%d ", matriz[i][j]);
            printf("\n");
        }
    }
}
```



Em C/C++ você precisa indicar o tamanho de todas as dimensões de uma matriz passada por parâmetro, exceto a dimensão mais à esquerda.

Exemplo com **todas** as dimensões da matriz:

```
1 void imprimirMatriz(int matriz[3][3], int n, int m)
2 {
3     for (int i = 0; i < n; ++i) {
4         for (int j = 0; j < m; ++j)
5             printf("%d ", matriz[i][j]);
6         printf("\n");
7     }
8 }
```



- Ao passar um **vetor simples** como parâmetro, não é necessário fornecer o seu tamanho na declaração da função.
- Quando o **vetor é multi-dimensional** a possibilidade de não informar o tamanho na declaração se restringe à primeira dimensão apenas.

```
void mostra_matriz(int mat[][10], int n_linhas) {  
    ...  
}
```



Em C/C++ você precisa indicar o tamanho de todas as dimensões de uma matriz passada por parâmetro, exceto a dimensão mais à esquerda.

```
1 void imprimirMatriz2(int matriz[][10], int n, int m)
2 {
3     for (int i = 0; i < n; ++i) {
4         for (int j = 0; j < m; ++j)
5             printf("%d ", matriz[i][j]);
6         printf("\n");
7     }
8 }
```

```
1 int main()
2 {
3     int A[10][10];
4     ...
5     imprimirMatriz2(A, 10, 10);
6     ...
7     return 0;
8 }
```



Mas... porquê todas as dimensões menos a mais à esquerda??

- Por conta da forma como matrizes são representadas na memória!
- As linhas são colocadas sequencialmente em um “**vetorção**”.
- Exemplo: seja a matriz  $3 \times 3$  a seguir

```
1  int matriz[3][3] = { { 10, 20, 30 },  
2                        { 40, 50, 60 },  
3                        { 70, 80, 90 } };
```

- Ela será representada na memória como um vetor de tamanho 9:

```
1  i          --->    0  0  0  1  1  1  2  2  2  
2  j          --->    0  1  2  0  1  2  0  1  2  
3  M[i][j]    ---> { 10, 20, 30, 40, 50, 60, 70, 80, 90 }
```



## Passando matrizes por parâmetro

Logo, quando acessamos o campo  $[i][j]$  de uma matriz  $4 \times 5$ :

- C/C++ acessa o campo  $5 \times i + j$  do “vetorzão” (em que 5 é o número de colunas).
- Para tal, o compilador deve saber quantas colunas há em cada linha
  - Ou seria impossível multiplicar por **5** neste exemplo.

Se você não quiser definir as dimensões da sua matriz em tempo de compilação, há algumas alternativas...

- que aprenderemos em breve...  
(quando falarmos sobre **alocação dinâmica**)



JESUÍTAS BRASIL



Somos infinitas possibilidades

- Pode-se criar uma função deixando de indicar a primeira dimensão:

```
void mostra_matriz(int mat[][10], int n_linhas) {  
    ...  
}
```

- Ou pode-se criar uma função indicando todas as dimensões:

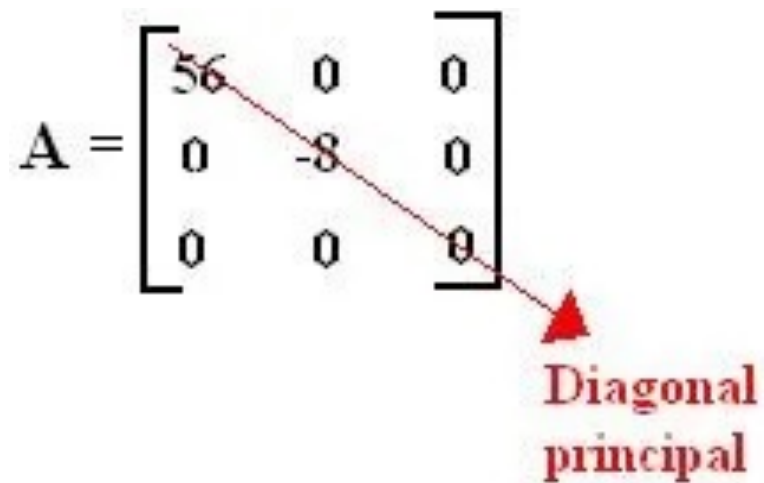
```
void mostra_matriz(int mat[5][10], int n_linhas) {  
    ...  
}
```

- Mas não pode-se deixar de indicar outras dimensões (exceto a primeira):

```
void mostra_matriz(int mat[5][], int n_linhas) {  
    //ESTE NÃO FUNCIONA  
    ...  
}
```



Exercício 4 – Faça um programa em C que implementa um função que recebe uma matriz e imprime a sua diagonal principal.

$$A = \begin{bmatrix} 56 & 0 & 0 \\ 0 & -8 & 0 \\ 0 & 0 & 0 \end{bmatrix}$$


Diagonal principal





# Propriedades de matrizes

Criar uma aplicação com operações básicas sobre matrizes quadradas:

- Soma de 2 matrizes com dimensões  $n \times n$ .
- Subtração de 2 matrizes com dimensões  $n \times n$ .
- Cálculo da transposta de uma matriz de dimensão  $n \times n$ .
- Multiplicação de 2 matrizes com dimensões  $n \times n$ .



JESUÍTAS BRASIL



Somos infinitas possibilidades

DESAFIO 1 ! Com uma matriz 3x3, estruturas de repetição e condicionais (if – else), como podemos implementar um jogo da velha em C? Implemente o jogo para 2 jogadores.



JESUÍTAS BRASIL



Somos infinitas possibilidades

DESAFIO 2 ! Faça um programa o qual determina se uma matriz é inversível. Para verificar basta calcular o determinante da matriz. Se o determinante for diferente de zero, então a matriz é inversível.



JESUÍTAS BRASIL



Somos infinitas possibilidades