# Matriz em C

# O que é uma Matriz?

- Em C, uma **matriz é um vetor multidimensional**, mais especificamente, **um vetor de vetores**.
  Ou seja, **um conjunto de valores organizados em linhas e colunas** — como se fosse uma **tabela**
- Você usa uma **matriz (lista de listas)** quando precisa **organizar dados em linhas e colunas**, como em:

| Problema real | Como uma matriz ajuda? |
|---|---|
| Tabela de preços, horários, ou notas | Cada linha pode ser um item ou pessoa, e as colunas os dados |
| Mapas ou jogos com tabuleiro | A matriz representa as casas do mapa/tabuleiro |
| Grades de horários escolares | Cada célula representa uma disciplina em um horário |
| Planilhas financeiras | Matriz imita o funcionamento de uma planilha Excel |
| Operações entre imagens (em processamento de imagem) | Imagem = matriz de pixels |

# Estrutura geral de uma matriz

```
tipo nome_matriz[linhas][colunas];
```

📌 **Exemplo:**

```c
int matriz[3][4];
```

Essa declaração cria uma matriz de **3 linhas e 4 colunas**, capaz de armazenar 12 números inteiros.

# Declarando, preenchendo e exibindo uma matriz 2x2

```c
#include <stdio.h>

int main() {
    int matriz[2][2];  // Declara uma matriz 2x2 do tipo inteiro

    // Preenchendo a matriz com valores informados pelo usuário
    for (int i = 0; i < 2; i++) {              // Loop para percorrer as linhas
        for (int j = 0; j < 2; j++) {          // Loop para percorrer as colunas
            printf("Digite o valor para [%d][%d]: ", i, j);
            scanf("%d", &matriz[i][j]);        // Lê o valor e armazena na posição correspond
        }
    }

    // Exibindo a matriz
    printf("\nMatriz digitada:\n");
    for (int i = 0; i < 2; i++) {              // Loop para percorrer as linhas novamente
        for (int j = 0; j < 2; j++) {          // Loop para colunas
            printf("%d\t", matriz[i][j]);      // Imprime o valor da posição i,j
        }
        printf("\n");                          // Quebra de linha após cada linha da matriz
    }

    return 0;
}
```

Explicação detalhada:

| Linha | Explicação |
|---|---|
| `int matriz[2][2];` | Declara uma matriz com 2 linhas e 2 colunas. Total de 4 posições. |
| Dois `for` aninhados | Usados para acessar cada elemento da matriz |
| `scanf("%d", &matriz[i][j]);` | Lê um valor inteiro do usuário e armazena na posição correspondente |
| `printf("%d\t", matriz[i][j]);` | Imprime os valores da matriz formatados em colunas com `\t` (tabulação) |

✅ `\t`

É um **caractere de escape** em C que significa **tabulação horizontal** — ou seja, ele **insere um espaço extra entre os elementos** como se você tivesse apertado a tecla **TAB** do teclado.

📌 É usado para deixar as saídas **alinhadas** em colunas.

---

✅ `%d\t` **junto:**

É como dizer:

> **"Mostre um número inteiro e depois pule um tab antes de continuar imprimindo."**

# Somando os elementos de uma matriz 2x2

```c
#include <stdio.h>

int main() {
    int matriz[2][2];
    int soma = 0;

    // Leitura da matriz
    for (int i = 0; i < 2; i++) {
        for (int j = 0; j < 2; j++) {
            printf("Digite o valor para [%d][%d]: ", i, j);
            scanf("%d", &matriz[i][j]);
            soma += matriz[i][j];  // Acumula a soma dos elementos
        }
    }

    // Exibe a soma
    printf("Soma de todos os elementos: %d\n", soma);

    return 0;
}
```

| Conceito | Explicação |
| --- | --- |
| `matriz[i][j]` | Acessa o valor da linha `i` e coluna `j` |
| Dois `for` aninhados | São essenciais para percorrer cada célula da matriz |
| Matrizes são fixas | Em C, o tamanho da matriz deve ser definido na declaração (estática) |

Exemplo de **matrizes com tamanho variável** (chamadas de **VLA – Variable Length Arrays**) **com tamanho definido em tempo de execução** (C99+):

Esses nomes se referem aos **padrões oficiais da linguagem C**, criados e atualizados por um órgão chamado **ISO** (Organização Internacional de Padronização).

Cada vez que o C recebe uma **atualização na linguagem**, com **novos recursos e melhorias**, é lançado um novo padrão com o nome do **ano de criação**.

✅ **Principais padrões da linguagem C:**

| Padrão | Ano | Também chamado de... | Destaques do padrão |
|---|---|---|---|
| C89 | 1989 | ANSI C | Primeiro padrão oficial do C |
| C90 | 1990 | ISO C | C89 aceito pela ISO |
| C99 | 1999 | C99 ou C99+ | Muitas melhorias: VLA, `//` comentários, `for(int i=0; ...)`, etc |
| C11 | 2011 | Modern C | Threads, `_Generic`, maior segurança |
| C17 | 2017 | Último estável | Correções menores (sem grandes novidades) |
| C23 | 2023 | Novo padrão moderno | Ainda ganhando suporte pelos compiladores |

```c
#include <stdio.h>

int main() {
    int linhas, colunas;

    // Usuário define o tamanho da matriz
    printf("Digite o número de linhas: ");
    scanf("%d", &linhas);
    printf("Digite o número de colunas: ");
    scanf("%d", &colunas);

    // Declarando uma matriz com tamanho variável (VLA - válido em C99+)
    int matriz[linhas][colunas];

    // Preenchendo a matriz
    for (int i = 0; i < linhas; i++) {
        for (int j = 0; j < colunas; j++) {
            printf("Digite o valor para [%d][%d]: ", i, j);
            scanf("%d", &matriz[i][j]);
        }
    }

    // Exibindo a matriz
    printf("\nMatriz digitada:\n");
    for (int i = 0; i < linhas; i++) {
        for (int j = 0; j < colunas; j++) {
            printf("%d\t", matriz[i][j]);
        }
        printf("\n");
    }

    return 0;
}
```

# Matrizes muito grandes ou mais flexíveis,

- VLA (**Variable Length Arrays)** ainda usa a pilha de memória, o que pode ser arriscado se os tamanhos forem grandes.
- Para matrizes muito grandes ou mais flexíveis, o ideal é usar alocação dinâmica com ponteiros e malloc() (nível mais avançado)

# O que é malloc()?

## ✅ Definição simples:

`malloc()` é uma **função da linguagem C** que **aloca (reserva) um espaço na memória** durante a execução do programa (ou seja, **em tempo de execução**).

> **malloc significa "memory allocation" (*alocação de memória*).**

## ✅ Sintaxe:

```c
ponteiro = (tipo *) malloc(tamanho_em_bytes);
```

### Explicando:

- `ponteiro` → recebe o endereço da memória alocada
- `(tipo *)` → faz a **conversão (cast)** do ponteiro para o tipo desejado
- `malloc(...)` → recebe a quantidade de **bytes** que você quer reservar

| Tipo de memória | Exemplo | Fica pronta quando? | Vantagem |
|---|---|---|---|
| Estática | `int x[10];` | Na **compilação** | Mais simples, mas fixa |
| Dinâmica | `malloc(...)` | Durante a **execução** | Flexível, uso de ponteiros |

## ⚠️ IMPORTANTE: sempre usar `free()`

Tudo que você aloca com `malloc()` **não é liberado automaticamente**.
Então, você deve **liberar manualmente** com `free()`:

```c
free(vetor);
```

## 📌 Exemplo básico:

```c
int *vetor;
vetor = (int *) malloc(5 * sizeof(int));
```

### O que está acontecendo aqui?

1. `sizeof(int)` → retorna quantos bytes ocupa um `int` (geralmente 4)
2. `5 * sizeof(int)` → quer reservar espaço para **5 inteiros**
3. `malloc(...)` → pede esse espaço na memória
4. `(int *)` → converte o resultado para um **ponteiro para int**
5. `vetor` → agora aponta para o **início dessa área de memória**

## 📌 Por que usar `malloc()`?

Porque em C, normalmente usamos:

```c
int vetor[5];  // Alocação estática (fixa)
```

Mas com `malloc()`, podemos fazer isso **dinamicamente**:

```c
int *vetor;
int tamanho;
scanf("%d", &tamanho);
vetor = (int *) malloc(tamanho * sizeof(int));
```

✅ Assim, o **usuário escolhe o tamanho**, e seu programa é **mais flexível**.

# Exemplo completo com malloc():

```c
#include <stdio.h>
#include <stdlib.h>  // Necessário para malloc() e free()

int main() {
    int linhas, colunas;

    // Solicita ao usuário o tamanho da matriz
    printf("Digite o número de linhas: ");
    scanf("%d", &linhas);

    printf("Digite o número de colunas: ");
    scanf("%d", &colunas);

    // ------------------------------
    // ALOCAÇÃO DINÂMICA DA MATRIZ
    // ------------------------------

    // Passo 1: Criar um vetor de ponteiros, cada um representando uma linha
    int **matriz = (int **) malloc(linhas * sizeof(int *));
    // Explicação:
    // - 'int **matriz' é um ponteiro para ponteiro (matriz)
    // - malloc(linhas * sizeof(int *)) aloca espaço para 'linhas' ponteiros
    // - Cada ponteiro representará uma linha da matriz

    // Passo 2: Para cada linha, alocamos um vetor de 'colunas' inteiros
    for (int i = 0; i < linhas; i++) {
        matriz[i] = (int *) malloc(colunas * sizeof(int));
        // Aqui estamos alocando espaço para 'colunas' inteiros em cada linha
    }

    // ------------------------------
    // PREENCHIMENTO DA MATRIZ
    // ------------------------------
    printf("\nDigite os elementos da matriz:\n");

    for (int i = 0; i < linhas; i++) {
        for (int j = 0; j < colunas; j++) {
            printf("Elemento [%d][%d]: ", i, j);
            scanf("%d", &matriz[i][j]);
            // Acessamos normalmente como matriz[i][j]
        }
    }

    // ------------------------------
    // EXIBIÇÃO DA MATRIZ
    // ------------------------------
    printf("\nMatriz digitada:\n");

    for (int i = 0; i < linhas; i++) {
        for (int j = 0; j < colunas; j++) {
            printf("%d\t", matriz[i][j]);  // \t para tabular os números
        }
        printf("\n");
    }

    // ------------------------------
    // LIBERAÇÃO DA MEMÓRIA
    // ------------------------------
    // Sempre que usamos malloc, precisamos liberar a memória com free()
    for (int i = 0; i < linhas; i++) {
        free(matriz[i]);  // Libera cada linha
    }
    free(matriz);  // Libera o vetor de ponteiros

    return 0;
}
```

# Exemplo 1: Boletim Escolar com Matriz Estática 3x3

```c
#include <stdio.h>

int main() {
    // Criamos uma matriz 3x3 do tipo float
    // Cada linha representa um aluno
    // Cada coluna representa uma matéria
    float notas[3][3];

    // --------------------------
    // Leitura dos dados (notas)
    // --------------------------
    for (int i = 0; i < 3; i++) {
        // i representa o número do aluno (linha da matriz)
        printf("\nAluno %d:\n", i + 1);

        for (int j = 0; j < 3; j++) {
            // j representa a matéria (coluna da matriz)
            printf("Digite a nota da matéria %d: ", j + 1);
            scanf("%f", &notas[i][j]);  // Armazena na posição [i][j]
        }
    }
```

| Loop | O que ele faz |
|---|---|
| `for (int i = 0; i < 3; i++)` | Percorre cada **linha da matriz** (cada aluno) |
| `for (int j = 0; j < 3; j++)` | Percorre cada **coluna da linha atual** (cada matéria) |

```c
    // ------------------------------
    // Exibindo o boletim (a matriz)
    // ------------------------------
    printf("\nBoletim completo:\n");

    for (int i = 0; i < 3; i++) {
        printf("Aluno %d: ", i + 1);

        for (int j = 0; j < 3; j++) {
            // Exibe cada nota do aluno i
            printf("%.1f ", notas[i][j]);
        }

        printf("\n");  // Quebra de linha ao final de cada aluno
    }

    // -------------------------------------
    // Calculando e exibindo a média por aluno
    // -------------------------------------
    printf("\nMédia de cada aluno:\n");

    for (int i = 0; i < 3; i++) {
        float soma = 0;

        for (int j = 0; j < 3; j++) {
            soma += notas[i][j];  // Soma todas as notas do aluno i
        }

        float media = soma / 3.0;
        printf("Aluno %d: %.2f\n", i + 1, media);
    }

    return 0;
}
```

# Exemplo 2 – Grade de Horários Escolar (Agenda Semanal)

```c
#include <stdio.h>
#include <string.h>

int main() {
    // Criamos uma matriz de strings [5][3], onde:
    // - 5 são os dias da semana (linhas)
    // - 3 são os horários/aulas por dia (colunas)
    char grade[5][3][30];  // 30 é o tamanho máximo de cada disciplina

    // Lista com os nomes dos dias para facilitar a exibição
    char dias[5][10] = {"Segunda", "Terça", "Quarta", "Quinta", "Sexta"};

    // ---------------------------
    // Leitura das disciplinas
    // ---------------------------
    for (int i = 0; i < 5; i++) {  // i representa o dia da semana (linha)
        printf("\nDia: %s\n", dias[i]);

        for (int j = 0; j < 3; j++) {  // j representa a aula (coluna)
            printf("Digite a disciplina da aula %d: ", j + 1);
            scanf(" %[^\n]", grade[i][j]);  // Lê uma string com espaços
        }
    }
```

```c
    // ---------------------------
    // Exibição da grade
    // ---------------------------
    printf("\nGrade de Horários:\n");

    for (int i = 0; i < 5; i++) {
        printf("\n%s:\n", dias[i]);

        for (int j = 0; j < 3; j++) {
            printf("  Aula %d: %s\n", j + 1, grade[i][j]);
        }
    }

    return 0;
}
```

### 🧠 O que é `#include <string.h>`?

Esse `#include` serve para importar a **biblioteca padrão de manipulação de strings** da linguagem C.

Ela traz várias **funções prontas** para você trabalhar com **cadeias de caracteres** (strings), como:

| Função | O que faz |
|---|---|
| `strcpy(dest, src)` | Copia uma string para outra |
| `strlen(str)` | Retorna o tamanho da string |
| `strcmp(a, b)` | Compara duas strings |
| `strcat(a, b)` | Concatena (junta) duas strings |

- Esse formato lê uma string **com espaços**, até o final da linha.
- Sem esse espaço antes do %[^\n], o scanf não funciona direito após usar enter.

# Exemplo 3 – Controle de Presença em Sala de Aula

```c
#include <stdio.h>     // Biblioteca para entrada e saída padrão
#include <ctype.h>     // Biblioteca para manipulação de caracteres (toupper)

int main() {
    // Vetor com nomes dos 3 alunos
    char alunos[3][20] = {"Ana", "Carlos", "Beatriz"};

    // Vetor com os 5 dias da semana
    char dias[5][10] = {"Seg", "Ter", "Qua", "Qui", "Sex"};

    // Matriz para armazenar as presenças: 3 linhas (alunos) x 5 colunas (dias)
    char presenca[3][5];
    // Leitura das presenças
    // ----------------------
    for (int i = 0; i < 3; i++) {
        // i representa o índice do aluno (linha)
        printf("\nPresenças para %s:\n", alunos[i]);

        for (int j = 0; j < 5; j++) {
            // j representa o índice do dia (coluna)
            char valor;

            do {
                printf("%s - Digite 'P' para Presente ou 'F' para Falta: ", dias[j]);
                scanf(" %c", &valor);      // Lê o caractere com espaço antes do %c para evit

                valor = toupper(valor);    // Converte o caractere para maiúsculo (ex: 'p' →

                if (valor == 'P' || valor == 'F') {
                    presenca[i][j] = valor;  // Armazena o valor válido na matriz
                } else {
                    printf("Entrada inválida! Use apenas P ou F.\n");
                }

            } while (valor != 'P' && valor != 'F');  // Repete até que a entrada seja válida
        }
    }

    // Exibição da tabela de presenças
    // -------------------------------
    printf("\n📋 Tabela de Presenças:\n");

    // Cabeçalho da tabela: imprime os dias
    printf("Aluno     | ");
    for (int j = 0; j < 5; j++) {
        printf("%s ", dias[j]);
    }
    printf("\n");

    // Corpo da tabela: mostra cada linha com nome + presenças
    for (int i = 0; i < 3; i++) {
        printf("%-10s| ", alunos[i]);  // Imprime nome do aluno alinhado à esquerda
        for (int j = 0; j < 5; j++) {
            printf(" %c  ", presenca[i][j]);  // Imprime cada presença ('P' ou 'F')
        }
        printf("\n");
    }

    return 0;  // Fim do programa
}
```

# Exercícios

- **1.** Crie uma matriz 2x2. Peça ao usuário para digitar os 4 valores da matriz. Ao final, exiba a matriz linha por linha.

**2.** Crie uma matriz 3x3 com valores digitados pelo usuário. Em seguida, calcule e exiba:

- A soma de todos os elementos
- A média dos elementos
- Os valores da diagonal principal
- **3.** Crie uma matriz onde o usuário irá digitar a presença (P) ou falta (F) de 4 alunos durante 5 dias da semana.
  Ao final, exiba uma tabela formatada com os dados digitados.

# Exercícios

- **4.** Simule um jogo da velha. Crie uma matriz 3x3 e permita que dois jogadores joguem alternadamente, escolhendo a linha e a coluna onde querem marcar (X ou O).
  Não permita sobrescrever posições já ocupadas e exiba o tabuleiro a cada jogada.
- **5.** Crie uma matriz de 3 alunos e 3 matérias (matriz 3x3) onde o usuário digita as notas.
  Depois, exiba a média de cada aluno e a média de cada matéria (coluna).