



# Programação Imperativa e Funcional

## Vetores e Matrizes

Diego Bezerra

dfb2@cesar.school



# Na aula anterior...



- Discussão sobre Estruturas de Decisão e Repetição em C
- Exercícios usando o Beecrowd

# Objetivos da aula



- Vetores unidimensionais
  - Definição
  - Declaração
  - Inicialização
  - Acesso
- Vetores multidimensionais
- Strings

# Vetores: Conceito

- Representa uma coleção **homogênea** de dados
- Todos os elementos são do **mesmo tipo**
  - Não é possível um vetor com uma string e um double
- Elementos podem ser de tipo **básico** ou **composto**
- Possuem tamanho fixo depois de criados

2	9	3	11	1
vetor[0]	vetor[1]	vetor[2]	vetor[3]	vetor[4]

# Vetores: Declaração



```
int a [10]; //declara um vetor de inteiros
double x [2]; //declara um vetor de double
char s [5]; //declara uma string (cadeia de caracteres)
Cliente clientes [5]; //declara vetor de um tipo estruturado
```

- O vetor foi alocado na memória
- O espaço reservado para um vetor é baseado na quantidade de elementos e no tipo declarado!

# Vetores: Declaração e Criação



- Possível declarar e não inicializar as posições no vetor
  - Elementos são inicializados com **valor default do tipo**
  - **Cuidado! Usar uma variável não inicializada pode gerar a leitura de lixo de memória.**
- Possível declarar e inicializar as posições no vetor

```
int a [4];  
int b [4] = {1, 2, 3, 4};  
int c [4] = {1, 2, 3}; // O que acontece?
```

Primeiro item: a[0]  
Último item: a[3]

# Vetores: Criação (1/2)



- O tamanho do vetor pode ser definido por uma **variável** ou **expressão numérica**
  - O tamanho **pode** ser definido de forma dinâmica
  - O tamanho **não pode** ser alterado

```
int tamanho;  
scanf("%d", &tamanho); //solicitando ao usuário o tamanho do  
vetor  
char vetor1 [tamanho];  
double vetor2 [tamanho*2];
```

## Vetores: Criação (2/2)



- Podemos usar uma constante para definir o tamanho de um vetor
- A diretiva **#define** é utilizada para definir macros
  - Toda vez que o compilador encontra o identificador definido, ele o substitui pelo valor ou expressão associado.
- Também podemos utilizar o modificador **const**

```
#define TAM_MAX 10

const int TAM_MAX2 = 10;

char vetor1 [TAM_MAX];
double vetor2 [TAM_MAX2];
```



# Vetores: Inicialização e Acesso



- Uma vez que o vetor foi criado, seus elementos podem ser acessados através do **índice**
  - Valor inteiro não-negativo
  - Intervalo **[0, n-1]**, sendo **n = total de elementos**

```
double salarios [5];  
salarios[0] = 2000.00;  
salarios[1] = 2500.00;  
salarios[2] = 4100.00;  
salarios[3] = 1450.00;  
salarios[4] = 6050.00;
```

# Vetores: Inicialização



- **Inicialização total:** todos os elementos são explicitamente inicializados
- **Inicialização parcial:** apenas alguns elementos são explicitamente inicializados e os restantes assumem o valor default
- **Boas práticas:**
  - Sempre que possível, inicializar completamente os arrays para garantir que os dados estejam em um estado conhecido.
  - Utilizar funções ou laços para garantir que todos os elementos sejam definidos, especialmente em arrays grandes.

```
int idades [5] = {19, 20, 18, 22, 21};  
double alturas [5] = {1.84, 1.71, 1.62};
```

# Vetores: Acesso fora dos limites



- Um comportamento indefinido pode ser gerado caso se tente acessar índices inválidos
  - Valor inteiro não-negativo
  - Fora do intervalo  $[0, n-1]$ , sendo  $n$  = total de elementos
- **Riscos:**
  - Leitura de lixo de memória, corrupção de dados e até crash do programa (segmentation fault)

```
double salarios [5];  
salarios[10] = 2000.00;  
salarios[-1] = 2000.00;
```

# Vetores: Inicialização a partir do teclado



- Passamos para a função **scanf** o endereço de cada elemento do vetor (**&idades[i]**)

```
int idades [3];  
  
scanf("%d", &idades[0]);  
scanf("%d", &idades[1]);  
scanf("%d", &idades[2]);
```

# Vetores: Imprimindo elementos do vetor

```
int idades [3];

idades[0] = 63;
idades[1] = 44;
idades[2] = 21;

printf("%d", idades[0]);
printf("%d", idades[5]); // O que acontece se fizermos acesso
a um elemento indefinido de um vetor?
```

```
for (int i = 0; i < 3; i++) {
    scanf("%d", &idades[i]);
}
for (int i = 0; i < 3; i++) {
    printf("%d", idades[i]);
}
```

Inicialização e impressão de um vetor usando laço **for**

# Vetores: Dica!



- C não possui uma função built-in para medir o tamanho dos vetores como `len()`, `size()` ou `length()`
- Como solução, usa-se uma constante ou mesmo uma variável para definir o tamanho

```
#define SIZE 3 // ou const int size =  
3;
```

```
int main() {  
    int idades[SIZE];  
    for (int i = 0; i < SIZE; i++) {  
        scanf("%d", &idades[i]);  
    }  
    return 0;  
}
```

```
int main() {  
    int size;  
    scanf("%d", &size);  
    int idades[size];  
    for (int i = 0; i < size; i++) {  
        scanf("%d", &idades[i]);  
    }  
    return 0;  
}
```

# Vetores Multidimensionais



- São vetores com **2** ou **mais** dimensões
- São úteis para representar **matrizes**
  - O primeiro valor entre colchetes representa a **linha** e o segundo, a **coluna**

	Coluna 0	Coluna 1	Coluna 2	Coluna 3
Linha 0	a[0][0]	a[0][1]	a[0][2]	a[0][3]
Linha 1	a[1][0]	a[1][1]	a[1][2]	a[1][3]
Linha 2	a[2][0]	a[2][1]	a[2][2]	a[2][3]

# Vetores Multidimensionais: Matrizes

- Declaração, criação e inicialização

```
int a [3][4]; // Multidimensional irregular
int b [2][2] = { {1,2}, {3,4} }; // Matriz quadrada
```

- Acesso

```
b[0][1] = 19;
b[1][0] = -1;

printf("%d", b[1][0]);

b[1][4] = 10; ❌
```

	0	1	
0	1	2	→
1	3	4	

	0	1
0	1	19
1	-1	4



# Matrizes: Manipulação



- Declarar, preencher uma matriz e imprimir uma matriz

```
#include <stdio.h>
#define LINHAS 2
#define COLUNAS 2
int main() {
    int matriz[LINHAS][COLUNAS];
    for (int i=0; i < LINHAS; i++) {
        for (int j = 0; j < COLUNAS; j++) {
            scanf("%d", &matriz[i][j]);
        }
    }
    for (int i=0; i < LINHAS; i++) {
        for (int j = 0; j < COLUNAS; j++) {
            printf("%d", matriz[i][j]);
        }
    }
    return 0;
}
```

# Matrizes: Exercícios



- Imprimir apenas os elementos da diagonal principal
- Imprimir apenas os elementos da primeira e última linha
- Imprimir apenas os elementos da última coluna
- Somar todos os elementos de uma matriz



# Exercícios revisão



**QUESTÃO 01** - Indique se as seguintes afirmações sobre variáveis, estruturas de decisão e repetição em C são verdadeiras (V) ou falsas (F). Quando uma afirmação for indicada como falsa, justifique.

1. ☐ Em C, uma variável deve ser declarada (existir) antes de ser utilizada em qualquer expressão ou função.
2. ☐ O tipo `char` em C pode ser usado para armazenar um caractere ou um pequeno número inteiro, e sua representação ocupa 2 bytes na memória.
3. ☐ Em uma estrutura `if-else` em C, é possível ter múltiplas instruções dentro de cada bloco sem a necessidade de usar chaves `{}`.
4. ☐ No comando `for` em C, todas as partes (inicialização, condição e incremento) são obrigatórias, e o loop não funciona se qualquer uma delas for omitida.

**QUESTÃO 02** - Escreva um programa em C que recebe duas notas e calcula a média ponderada, cujos pesos são 7.0 e 3.0 de dez alunos. Seu programa deve utilizar matrizes e as estruturas de controle vistas até agora e atender às instruções a seguir:

- A. Pedir que o usuário informe as duas notas;
- B. Realizar a operação para calcular a média.
- C. Imprimir uma mensagem indicando se cada aluno foi aprovado (caso a média seja maior ou igual a 5) ou reprovado (caso contrário).

# Referências



- Rangel Netto, J. L. M., Cerqueira, R. D. G., & Celes Filho, W. (2004). **Introdução a estrutura de dados: com técnicas de programação em C.**