

# Linguagem de Programação I - ESP201

Prof<sup>a</sup> Ana Carolina Sokolonski

Bacharelado em Sistemas de Informação  
Instituto Federal de Ciência e Tecnologia da Bahia  
Campus de Feira de Santana

*carolsoko@ifba.edu.br*

February 11, 2025

# Estruturas de Dados Homogêneas e Heterogêneas

- 1 Estruturas de Dados Homogêneas Unidimensionais - Vetores
- 2 Estruturas de Dados Homogêneas Bidimensionais - Matrizes
- 3 Estruturas de Dados Heterogêneas - Estruturas (Structs)
- 4 Referências

# Estruturas de Dados Homogêneas e Heterogêneas

## Estruturas de Dados Homogêneas

Tipo de dado usado para representar uma coleção de variáveis de um mesmo tipo. Ex: Armazenar as idades de 100 pessoas ...

```
#include <stdio.h>
void main (){
int idade1, idade2, idade3, idade4, ..., idade100;
    printf ("Informe a idade da primeira pessoa: ");
    scanf("%d", &idade1);
    printf ("Informe a idade da segunda pessoa: ");
    scanf("%d", &idade2);
    :::
    printf ("Informe a idade da centésima pessoa: ");
    scanf("%d", &idade100);
}
```

## Estruturas de Dados Homogêneas

As Estruturas de Dados Homogêneas permitem que um mesmo tipo de dado seja armazenado num local da memória do computador e seja referenciado por um nome comum. Um elemento específico em uma estrutura de dados homogênea é acessado por meio de um índice. Em C, todas as estruturas de dados homogêneas consistem em posições contíguas na memória. O endereço mais baixo corresponde ao primeiro elemento e o mais alto, ao último elemento.[Schildt e Mayer 1997].

Na linguagem C, existem três tipos de Estruturas de Dados Homogêneas:

- 1 Unidimensionais - Vetores
- 2 Bidimensionais - Matrizes
- 3 Multidimensionais - Matrizes de mais de 2 dimensões - não estudaremos essas devido à complexidade e pouco uso

# Estruturas de Dados Homogêneas Unidimensionais Vetores

# Vetores

**Questão:** Qual a diferença entre encontrar uma palavra em uma lista de 10 palavras e encontrar esta palavra em uma lista de 10000 palavras? O tempo de processamento é apenas dez vezes maior? É mil vezes maior? Ou é um milhão de vezes maior?

# Vetores



# Vetores

## Definição:

Um *vetor*, ou *arranjo (array)*, é uma estrutura de dados que armazena uma sequência de objetos, todos do mesmo tipo, em posições consecutivas da memória RAM (*random access memory*) do computador. Essa estrutura permite acesso aleatório: qualquer elemento do vetor pode ser alcançado diretamente, sem passar pelos elementos anteriores (o décimo elemento, por exemplo, pode ser alcançado sem que seja necessário passar antes pelo primeiro, segundo, etc., nono elementos). [Cormen et al. 2009]

# Vetores

Como outras variáveis, como a linguagem C é fortemente tipada, os **Vetores (arrays)** devem ser explicitamente declarados para que o compilador possa alocar espaço para eles na memória.

Deve-se declarar o tipo do dado do vetor, que é o tipo de cada elemento do vetor, a quantidade máxima de elementos que o vetor será capaz de guardar, e o nome que essa coleção, ou seja, esse vetor, receberá.

*tipoDoDado*nomeVetor[tamanho];

- Vetor de 10 números inteiros: `int idade[10];`
- Vetor de 30 números reais: `double notas[30];`
- Vetor de 40 caracteres: `char letras[40];`

# Vetores

Suponha que deseja-se armazenar  $N$  números inteiros num vetor  $V$ . O espaço ocupado pelo vetor na memória pode ser criado pela declaração:

```
int v[N];
```

# Vetores

Suponha que deseja-se armazenar  $N$  números inteiros num vetor  $V$ . O espaço ocupado pelo vetor na memória pode ser criado pela declaração:

```
int v[N];
```

sendo  $N$  uma constante. Considerando que os números são armazenados nas posições de 0 a  $(n - 1)$ , então:

$v[0..(n - 1)]$  é um vetor de inteiros.

Claro que  $0 \leq n \leq N$ . Se  $n == 0$ , o vetor  $v[0..(n - 1)]$  está vazio. Se  $n == N$ , o vetor está cheio.

# Vetores

```
#include <stdio.h>
#include <stdlib.h>

int busca (int x, int n, int v[]){
    int i;
    i = n-1;
    while (i >= 0 && v[i] != x)
        i -= 1;
    return i;
}

int main (void){
    int i,j,aux;
    int v1[10],v[]={8,3,4,7,6,5,1,2,9,0};

    for (i = 0; i < 10; ++i){
        printf("Vetor original v[%d] = %d\n", i, v[i]);
        v1[v[i]] = i; } //permutando aqui

    for (j = 0; j < 10; ++j) printf("Permutado v[%d] = %d\n", j, v1[j]);

    printf("Informe qual o item deseja encontrar no vetor original:");
    scanf("%d",&aux);

    i = busca(aux, 10, v);
    printf("Posição do item = %d\n", i);
}
```

# Estruturas de Dados Homogêneas Bidimensionais Matrizes

# Matrizes

Como outras variáveis, como a linguagem C é fortemente tipada, as **Matrizes** devem ser explicitamente declaradas para que o compilador possa alocar espaço para elas na memória.

Deve-se declarar o tipo do dado da matriz, que é o tipo de cada elemento da matriz, a quantidade máxima de elementos que a matriz será capaz de guardar, e o nome que essa coleção, ou seja, essa matriz, receberá.

```
tipoDoDadonomeMatriz[tamanho1][tamanho2];
```

- Matriz quadrada de 10X10 inteiros: `int notasAlunos[10][10];`
- Matriz de 10X30 reais: `double numeros[10][30];`
- Matriz de 5X40 caracteres: `char letras[5][40];`

# Matrizes

Exemplo do uso de Matrizes: Numa sala de aula, possuímos 20 alunos e precisamos armazenar suas 4 notas, então faremos uma tabela:



# Matrizes

Exemplo do uso de Matrizes: Numa sala de aula, possuímos 20 alunos e precisamos armazenar suas 4 notas, então faremos uma tabela:

Aluno	Prova1	Prova2	Prova3	Prova4
José	2,0	8,0	6,0	10,0
Liz	3,0	9,0	2,0	2,3
Rosa	6,0	6,0	6,0	6,0
Igor	5,0	7,0	9,0	8,0
...	...	...	...	...
Sara	4,0	2,0	6,0	2,0

# Matrizes

Para representar essa tabela, devemos atribuir a cada aluno um índice, como a matriz é homogênea, só permite um mesmo tipo, só poderemos armazenar as notas, então os nomes não serão armazenados, os alunos serão identificados pelo índice. Dica: ordenar a lista em ordem alfabética para que seja fácil usar os índices de forma direta, o menor índice será o menor nome na ordem alfabética. A declaração da matriz fica assim:

# Matrizes

Para representar essa tabela, devemos atribuir a cada aluno um índice, como a matriz é homogênea, só permite um mesmo tipo, só poderemos armazenar as notas, então os nomes não serão armazenados, os alunos serão identificados pelo índice. Dica: ordenar a lista em ordem alfabética para que seja fácil usar os índices de forma direta, o menor índice será o menor nome na ordem alfabética. A declaração da matriz fica assim:

```
float notasAlunos[20][4];
```

# Matrizes

```
9  #include <stdio.h>
10 #define tamanho 5
11
12 void main(){
13     double notaAluno[tamanho][4];
14     for (int i = 0; i <= (tamanho-1); i++){
15         for (int j = 0; j <= 3; j++){
16             printf ("Informe Prova %d da %dª pessoa: ",j+1,i+1);
17             scanf ("%lf", &notaAluno[i][j]);
18         }
19     }
20
21     printf("Segue listagem de notas por aluno: \n");
22     for (int i = 0; i <= (tamanho-1); i++){
23         printf ("%dº aluno: ",i+1);
24         for (int j = 0; j <= 3; j++){
25             printf("%2.2f ",notaAluno[i][j]);
26         }
27         printf ("\n");
28     }
29 }
```

# Matrizes

**Exercício:** A distância entre várias cidades é dada pela tabela abaixo (em km):

x/y	1	2	3	4	5
1	00	15	30	05	12
2	15	00	10	17	28
3	30	10	00	03	11
4	05	17	03	00	80
5	12	28	11	80	00

**Table:** Distâncias entre as cidades (x,y)

Construa um programa que leia a tabela acima e informe ao usuário a distância entre duas cidades (x,y) por ele escolhidas, até que ele entre com o código 0 para qualquer uma das cidades;

# Matrizes

```
#include <stdio.h>

void main(){
    int distancias[5][5];
    int c1,c2,i,j;

    for(i=0; i<=4; i++) distancias[i][i] = 0;
    distancias[0][1] = distancias[1][0] = 15;
    distancias[0][2] = distancias[2][0] = 30;
    distancias[0][3] = distancias[3][0] = 5;
    distancias[0][4] = distancias[4][0] = 12;
    distancias[1][2] = distancias[2][1] = 10;
    distancias[1][3] = distancias[3][1] = 17;
    distancias[1][4] = distancias[4][1] = 28;
    distancias[2][3] = distancias[3][2] = 03;
    distancias[2][4] = distancias[4][2] = 11;
    distancias[3][4] = distancias[4][3] = 80;

    do{
        printf("Informe as cidades que deseja buscar:");
        scanf("%d %d",&c1,&c2);

        if((c1!=0)&&(c2!=0))
            printf("A distância entre as cidades é %d\n",distancias[c1-1][c2-1]);
    }while((c1!=0)&&(c2!=0));
}
```

# Matrizes

As matrizes também podem ser inicializadas na declaração:

```
float mat[4][3] = {{1,2,3},{4,5,6},{7,8,9},{10,11,12}};
```

Ou podemos inicializar sequencialmente:

```
float mat[4][3] = {1,2,3,4,5,6,7,8,9,10,11,12};
```

O número de elementos por linha pode ser omitido numa inicialização, mas o número de colunas deve, obrigatoriamente, ser fornecido:

```
float mat[ ][3] = {1,2,3,4,5,6,7,8,9,10,11,12};
```

# Matrizes

```
#include <stdio.h>

void main(){
    int distancias[5][5] = {{00,15,30,05,12},
                             {15,00,10,17,28},
                             {30,10,00,03,11},
                             {05,17,03,00,80},
                             {12,28,11,80,00}};

    int c1,c2,i,j;

    do{
        printf("Informe as cidades que deseja buscar:");
        scanf("%d %d",&c1,&c2);

        if((c1!=0)&&(c2!=0))
            printf("A distância entre as cidades é %d\n",distancias[c1-1][c2-1]);
    }while((c1!=0)&&(c2!=0));
}
```



# Estruturas de Dados Heterogêneas

## Estruturas (**Structs**)

# Estruturas de Dados Heterogêneas - Estruturas (Struct)

Tipo de dado usado para representar uma coleção de variáveis de tipos diferentes. Ex: Armazenar os dados de cadastro de 100 pessoas matriculadas em uma academia... para **CADA** aluno da academia teríamos o seguinte algoritmo:

# Estruturas de Dados Heterogêneas - Estruturas (Struct)

## Algoritmo para **UM** aluno

```
#include <stdio.h>
void main () {
    int idade; float altura, peso;
    char nome[30], endereco[50], telefone[11];
    printf ("Informe o nome da pessoa: ");
    scanf ("%s", nome); getchar();
    printf ("Informe a idade, a altura e o peso pessoa: ");
    scanf ("%d, %f, %f", &idade, &altura, &peso);
    printf ("Informe o endereço da pessoa: ");
    scanf ("%s", endereco); getchar();
    printf ("Informe o telefone da pessoa: ");
    scanf ("%s", telefone); getchar();
}
```

# Estruturas de Dados Heterogêneas - Estruturas (Struct)

## Algoritmo para 100 alunos

```
#include <stdio.h>
void main (){
int idade[100]; float altura[100], peso[100];
char nome[100][30], endereco[100][50], telefone[100][11];
for(int i = 0; i <= 99; i++){
    printf ("Informe o nome da pessoa: ");
    scanf ("%s", nome[i]); getchar();
    printf ("Informe a idade, a altura e o peso da pessoa: ");
    scanf ("%d,%f,%f",&idade[i],&altura[i],&peso[i]);
    printf ("Informe o endereço da pessoa: ");
    scanf ("%s", endereco[i]); getchar();
    printf ("Informe o telefone da pessoa: ");
    scanf ("%s", telefone[i]); getchar();
}}
```

## Estruturas de Dados Heterogêneas - Estruturas (Struct)

As Estruturas de Dados Heterogêneas permitem que vários tipos de dados diferentes sejam armazenados juntos, fazendo referência a um mesmo "DONO", ou seja, diversos tipos de dados diferentes são referenciados por um nome em comum.

Em C, pode-se fazer inclusive coleções (Matrizes e Vetores) de *structs*. [Schildt e Mayer 1997]

## Estruturas de Dados Heterogêneas - Estruturas (Struct)

Assim como as demais variáveis da linguagem C, que é fortemente tipada, as **Estruturas (Structs)** devem ser explicitamente declarados para que o compilador possa alocar o espaço de memória para eles. Deve-se declarar os tipos dos dados de cada elemento do Struct: Assim, o tipo de dado aluno ficaria...

```
typedef struct alunoAcademia{  
    char nome[30];  
    int idade;  
    double altura;  
    double peso;  
    char endereco[50];  
    char telefone[11];  
} aluno;
```

## Estruturas de Dados Heterogêneas - Estruturas (Struct)

Dentro do **main** da linguagem C, para declarar **UM** aluno faríamos:

```
void main(){  
    aluno maria;  
    maria.nome = "Maria Antônia";  
    maria.idade = 23;  
    maria.peso = 56.8;  
    ...  
}
```

## Estruturas de Dados Heterogêneas - Estruturas (Struct)


Dentro do **main** da linguagem C, para declarar **100** alunos faríamos:

```
void main(){
aluno classe[100];
for(int i = 0; i <= 99; i++){
    printf ("Informe o nome da pessoa: ");
    scanf ("%s", classe[0].nome); getchar();
    printf ("Informe a idade, a altura e o peso da pessoa: ");
    scanf ("%d, %f, %f",
    &classe[0].idade, &classe[0].altura, &classe[0].peso);
    printf ("Informe o endereço da pessoa: ");
    scanf ("%s", classe[0].endereco); getchar();
    printf ("Informe o telefone da pessoa: ");
    scanf ("%s", classe[0].telefone); getchar();
}}
```



# Referências

# Referências

 CORMEN, T. H. et al. *Introduction to Algorithms*. 2nd. ed. [S.I.]: The MIT Press, 2009. ISBN 0262032937.

 SCHILDT, H.; MAYER, R. *C completo e total*. [S.I.]: Pearson University, 1997. ISBN 9788534605953.