



Linguagem C – Vetores e Ponteiros



Objetivos da Aula:

- **Vetores**
- **Ponteiros**
- **Estruturas**
- **Exercícios de fixação**



VETORES

- *Vetores são uma coleção de variáveis do mesmo tipo que são referenciadas pelo mesmo nome.*
- *Em C, um vetor consiste em localidades contíguas de memória.*
- *O elemento mais baixo corresponde ao primeiro elemento e o mais alto ao último.*
- *O vetor mais utilizado é o de caracteres.*

DECLARAÇÃO DE VETORES

A forma geral da declaração de um vetor é:

tipo nome_var[tamanho];

Onde:

- **tipo** é o tipo base do vetor e
- **tamanho** é a quantidade de elementos que o vetor conterá.

ACESSANDO UM VETOR

- *Os vetores são acessados através de índices colocados entre colchetes.*
- *O índice do primeiro elemento do vetor é 0 (ZERO).*
- **EXEMPLOS:**

```
int amostra[10]; /* vetor de 10 inteiros */  
amostra[0] = 2; /* primeiro elemento */  
amostra[9] = 7; /* último elemento
```

EXEMPLO DE APLICAÇÃO:

main() // o que faz esta função?

```
{  
    int x[10]; /* vetor com 10 elementos int */  
    int t;  
    for (t = 0; t < 10; t ++)  
        x [ t ] = t;  
}
```

LIMITES DE VETORES

- ***C não faz checagem dos limites dos vetores, isto é responsabilidade do programador. Logo, o código a seguir não causará nenhum erro.***

```
int elementos[10];
```

```
elementos[12] = 0;
```

```
elementos[10] = 0;
```



LIMITES DE VETORES

- *Uma string é por definição, um vetor de caracteres terminado em 0.*
- *Então, para declarar a string, devemos declarar sempre um elemento a mais para o terminador.*

LIMITES DE VETORES

Exemplo:

char mensagem[] = “Exemplo”

Ficará armazenado na memória como:

E	x	e	m	p	l	o	0
---	---	---	---	---	---	---	---

MATRIZES BIDIMENSIONAIS

- *C permite que sejam declaradas matrizes bidimensionais.*

- *Forma da declaração:*

tipo nome_var[dimensão1][dimensão2];

- *Exemplo:*

char tabela[5][5];

MATRIZES MULTIDIMENSIONAIS

- *De forma semelhante as matrizes bidimensionais, declaramos as multidimensionais. Veja por exemplo uma matriz de 4 dimensões:*

```
int matriz[5][7][3][8];
```

INICIALIZAÇÃO DE MATRIZES

- *C permite que as matrizes globais sejam inicializadas.*

A forma geral é:

tipo nome_matriz[tam1]...[tamN] = {lista de valores}

Exemplo:

int i[10] = { 1, 2, 3, 4, 5, 6, 7, 8, 9, 10};

INICIALIZAÇÃO DE MATRIZES

```
int quadrados[5][2] = {  
    1, 1,  
    2, 4,  
    3, 9,  
    4, 16,  
    5, 25};
```

INICIALIZAÇÃO DE MATRIZES

- *CUIDADO COM OS VALORES A SEREM INICIALIZADOS: SE DECLARAR 10 ELEMENTOS, E INICIALIZAR TODOS OS ELEMENTOS, O COMPILADOR NÃO DEIXA ESPAÇO PARA O “ZERO”.*

INICIALIZAÇÃO DE MATRIZES

- *Podem ser declarados vetores sem especificar explicitamente seus tamanhos. Então, os vetores devem ser inicializados na declaração. O tamanho será definido na inicialização.*

Exemplo:

char mensagem[] = “Esta é uma string”;

Acessando os elementos das matrizes multidimensionais

main() // O que faz esta função?

```
{  
int numeros[4][3], i, j;  
for (i = 0; i < 4; i ++)  
    for (j = 0; j < 3; j++)  
        numeros[ i ][ j ] = i * j;  
}
```




PONTEIROS

- *Entender e usar corretamente os ponteiros são pontos cruciais para criação de programas bem-sucedidos em C.*
- *Além dos ponteiros serem uma das características mais fortes em C, também é a mais perigosa.*
- *É muito fácil usar ponteiros incorretamente causando erros difíceis de serem encontrados.*



PONTEIROS SÃO ENDEREÇOS

- *Um ponteiro é uma variável que contém um endereço de memória. Isto é, eles armazenam a localização (endereço) de outra variável dentro da memória do computador.*
- *Então dizemos que um ponteiro aponta para esta variável.*

DECLARAÇÃO DE PONTEIROS

- *A declaração de variáveis ponteiros, segue a seguinte regra geral:*

*tipo *nome_var;*

onde tipo é o tipo do elemento para o qual o ponteiro apontará.

Exemplo:

*char *p;*

*int *temp, *valor;*

OPERADORES DE PONTEIROS

- *Existem 2 operadores especiais de ponteiros: & e *.*
- *O operador & devolve o endereço da variável. É utilizado para fazer um ponteiro apontar para ela.*
- *O operador * devolve o valor armazenado no endereço apontado pelo ponteiro.*

EXEMPLOS DE PONTEIROS

main()

*{ int numero = 5, *p;*

p = №

// Qual o conteúdo de p?

**p = 3;*

// Qual o conteúdo de p?

numero = 7;

// Qual o conteúdo de p?

}



EXPRESSÕES COM PONTEIROS

- *C permite que sejam feitas expressões com ponteiros e elas seguem as mesmas regras das outras expressões em C.*
- *Quando se compara um ponteiro com outro, estamos comparando seus endereços. Isto é útil quando ambos os ponteiros apontam para elementos de um vetor.*

EXPRESSÕES COM PONTEIROS

- *Existe um relacionamento muito próximo entre os ponteiros e os vetores.*

- *Veja o código:*

```
char str[80], *p;
```

```
p = str;
```

*Este código faz com que p aponte para o primeiro elemento do vetor, pois **um vetor sem o índice se comporta como um ponteiro para seu primeiro elemento.***

EXEMPLOS:

- *Após a definição:*

*char str[80], *p;*

p = str;

são equivalentes os acessos ao quinto elemento de str:

str[4]

**(p + 4)*

p[4]



PROBLEMAS COM PONTEIROS:

- *É muito fácil errar quando se trabalha com ponteiros em C.*
- *Algumas vezes, os erros com ponteiros só aparecem quando o programa cresce.*
- *Ponteiros que não foram inicializados, apontam para um lugar desconhecido na memória, que pode ser inclusive o código do programa.*

PROBLEMAS COM PONTEIROS:

/ este programa está errado */*

main()

{

*int x, *p;*

x = 10;

**p = x;*

}

PROBLEMAS COM PONTEIROS:

/ este programa está errado */*

main()

{

*int x, *p;*

x = 10;

p = x;

}

VETORES DE PONTEIROS:

- Podemos construir vetores de ponteiros como declaramos vetores de qualquer outro tipo. Uma declaração de um vetor de ponteiros inteiros poderia ser:

```
int *pmatrx [10];
```

- No caso acima, **pmatrx** é um vetor que armazena **10 ponteiros para inteiros**.

ESTRUTURAS:

- *Uma estrutura é uma coleção de variáveis que são referenciadas pelo mesmo nome.*
- *É uma forma conveniente de manter juntas informações relacionadas.*
- *Forma geral:*

```
struct nome_estrutura {  
    tipo1 var1;  
    tipo2 var2;  
} var_estrutura;
```

Declarando variáveis do tipo estrutura

- *Além de poder declarar variáveis do tipo da estrutura durante a definição da estrutura, elas podem ser declaradas da seguinte forma:*

struct nome_estrutura nome_variável;

Acessando variáveis do tipo estrutura

- *Para acessar variáveis do tipo estrutura, utiliza-se o operador . (ponto).*
- *Forma geral:*
nome_variavel.nome_elemento;

Exemplo

```
struct pessoa {  
    char nome[21];  
    int idade;  
} primeiro;
```

```
main() {  
    struct pessoa segundo;  
    primeiro.idade = 20;  
    segundo = primeiro;  
}
```


Vetores e matrizes de estruturas

- *Podem ser declarados vetores e matrizes de estruturas, para isto, usamos a forma geral:*

struct nome_estrutura nome_var[t1][t2]...[tn];

- *Exemplo:*

struct pessoa pessoas[5];

Ponteiros para estruturas

- *Em C, podem ser declarados ponteiros para estruturas.*

- *Forma geral:*


*struct pessoa *primeiro;*

Ponteiros para estruturas

- *Em C, podem ser declarados ponteiros para estruturas.*

- *Forma geral:*

*struct pessoa *primeiro;*



Vantagens de se usar ponteiros para estruturas

- *Fazer chamada por referência para uma função;*
- *É mais rápido passar estruturas grandes por referência (usando ponteiros) do que por valor, pois estamos passando apenas um endereço.*

Acessando os elementos usando ponteiros para estruturas

- *Veja a declaração:*

```
struct pessoa *primeiro, segundo;  
segundo.idade = 10;  
primeiro = &segundo;
```

- *Para acessar o campo idade de primeiro:*

```
(*primeiro).idade // ou  
primeiro -> idade
```

Vetores, matrizes e estruturas dentro de estruturas

- *Os elementos das estruturas podem ser simples ou complexos, assim, podemos colocar vetores, matrizes e até estruturas dentro das estruturas. Veja:*

```
struct complexa {  
    char setor[21];  
    struct pessoa funcionarios[50];  
}
```

Exercício de Aplicação

- Escreva um programa em C que testa 5 nomes de um vetor (*array*) que contém suas respectivas idades e ache o de maior e menor idade.
- Dados: Maria tem 10 anos
Jose tem 5 anos
Pedro tem 4 anos
Sergio tem 7 anos
Carla tem 3 anos

Exercício de Aplicação

■ Resolução:

```
typedef struct{
```

```
    char nome[20];
```

```
    int idade;
```

```
}pessoa;
```

```
pessoa pessoas[5] = {"maria", 10, "jose", 5, "pedro", 4,  
    "sergio", 7, "carla",3};
```


Exercício de Aplicação

Resolução:

```
// mais_velho
pessoa * mais_velho(pessoa *p1, pessoa *p2){
    if(p1->idade < p2->idade)
        return p2;
    else
        return p1;
}

//mais_novo
pessoa *mais_novo (pessoa* p1, pessoa *p2){
    if(p1->idade < p2->idade)
        return p1;
    else
        return p2;
}
```



Exercício de Aplicação

Resolução:

```
main()
```

```
{
```

```
    pessoa *velho, *novo;
```

```
    int i;
```

```
    velho = novo = pessoas; // inicializa ponteiros
```

```
    for ( i =1; i < 5; i++)
```

```
    {
```

```
        velho = mais_velho(velho, &pessoas[i]);
```

```
        novo = mais_novo(novo, &pessoas[i]);
```

```
    }
```

```
}
```