

# ESTRUTURA DE DADOS

## AULA 06 – PONTEIROS (APONTADORES)

PARTE III



# VETOR DE PONTEIROS: DECLARAÇÃO

- Um vetor de ponteiros é feita de forma **similar** à **declaração** de **vetores** de qualquer outro tipo.

- Sintaxe:

*<tipo\_de\_dado> \*<nome\_array>[tamanho];*

- Ex.:

*int \*x[10]; //vetor que armazena 10 ponteiros para inteiros.*

```
int    *vet_ap[5];  
char   *vet_cadeias[5];
```

- O operador asterisco **\*** tem **precedência menor** que o de **indexação** **[]**.
- Vetores de ponteiros são tradicionalmente utilizados para:
  - **mensagens de erro**, que são constantes;
  - **ponteiros para strings**, pois uma *string* é essencialmente um ponteiro para o seu primeiro caractere.

# VETOR DE PONTEIROS

- Cada **posição** de um *array* de ponteiros pode **armazenar** o **endereço** de uma **variável** ou o **endereço** da posição **inicial** de um **outro array**.
- A manipulação de um *array* de ponteiros é **similar** à manipulação de um ponteiro. Como o **array** é **sempre indexado**, para atribuir o endereço de uma variável *x* a uma posição do *array* de ponteiros, **escreve-se**:  $p[indice] = \&x;$       *//sendo p ponteiro de tipo consistente com o tipo de dado de x;*
- Para **retornar** o **conteúdo** guardado pelo **ponteiro** nessa **posição** de memória, usa-se novamente o **\***:
  - $*p[indice]$
- **Ex.:**
  - $x[2]=\&var;$  *//sendo int \*x[5]; atribui o endereço de uma variável inteira chamada var ao terceiro elemento do vetor de ponteiros*
- **Verificando o conteúdo de var:**  $*x[2]$

## Vetor de Ponteiros: exemplo

```
1 #include <stdio.h>
2 #include <stdlib.h>
3 int main () {
4     int *pvet[2];
5     int x = 10, y[2] = {20,30};
6     pvet[0] = &x;
7     pvet[1] = y;
8     //imprime os endereços das variáveis
9     printf( "Endereco pvet[0]: %p\n", pvet[0] );
10    printf( "Endereco pvet[1]: %p\n", pvet[1] );
11    //imprime o conteúdo de uma variável
12    printf( "Conteudo em pvet[0]: %d\n", *pvet[0] )
13    ;
14    //imprime uma posição do vetor
15    printf( "Conteudo pvet[1][1]: %d\n", pvet
16            [1][1] );
17    system( "pause" );
18    return 0;
19 }
```

## Vetor de Ponteiros para *strings*

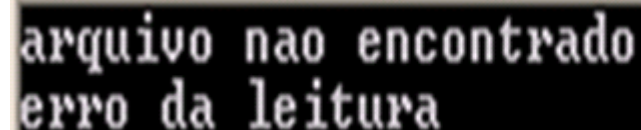
- ✓ Ponteiros para *strings*: uma *string* é essencialmente um ponteiro para o seu primeiro caractere.

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

int main()
{
    char *erro[2];           //ponteiro de vetor
    int i;
    erro[0]="arquivo nao encontrado\n"; //1ª posição
    erro[1]="erro da leitura\n";        //2ª posição

    for (i=0;i<2;i++)
    {
        printf("%s",erro[i]);
    }

    system("pause");
    return 0;
}
```



```
arquivo nao encontrado
erro da leitura
```

## Vetor de Ponteiros: exemplo

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

int main(void) {

    int i;
    char *erro[] = {
        "Arquivo nao pode ser aberto\n",
        "Erro de leitura\n",
        "Erro de escrita\n",
        "Falha de midia\n"};

    for (i=0;i<4;i++){
        printf("%s", erro[i]);
    }

    system("pause");
    return 0;
}
```

```
Arquivo nao pode ser aberto
Erro de leitura
Erro de escrita
Falha de midia
```

# PONTEIRO PARA ESTRUTURA

- Pode-se usar um **ponteiro** para **guardar o endereço** de um registro (***struct***), neste caso o **ponteiro aponta** para o **registro**.
- Lembre-se que cada **campo** do **registro** tem um **endereço** na memória do computador.
- O endereço da ***struct*** é o endereço de seu **primeiro campo**, de modo análogo ao ***array***.
- A declaração de um ponteiro para estrutura é feito de modo semelhante a outras declarações. Ex.:

```
struct dma {  
    int dia;  
    int mes;  
    int ano;  
};  
  
struct dma *p;    /* p é um ponteiro para registros dma */
```

## ○ Ponteiro para Estrutura: exemplo

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

```
struct dma {
    int dia;
    int mes;
    int ano;
};
```

```
main() {
```

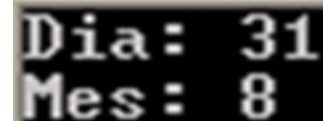
```
    struct dma *p;    /* p é um ponteiro para registros dma */
    struct dma x;
    p = &x;            /* agora p aponta para x */
    (*p).dia = 31;      /* mesmo efeito que x.dia = 31 */
    (*p).mes = 8;       /* mesmo efeito que x.mes = 8 */
```

```
    printf("Dia: %d\n", p->dia);
    printf("Mes: %d\n", (*p).mes);
```

```
    system("pause");
```

```
}
```

**É possível utilizar duas sintaxes que são equivalentes.**



```
Dia: 31
Mes: 8
```



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

```
struct minha_estrutura{
    int i;
    double f;
};
```

```
main() {
```

```
    struct minha_estrutura *p_minha_estrutura;
    struct minha_estrutura valor;
```

```
    p_minha_estrutura=&valor;
```

```
    p_minha_estrutura->i = 1;
```

```
    p_minha_estrutura->f = 1.2;
```

```
    printf("Valor de i: %d\n", p_minha_estrutura->i);
```

```
    printf("Valor de f: %f\n", p_minha_estrutura->f);
```

```
    (*p_minha_estrutura).i = 3;
```

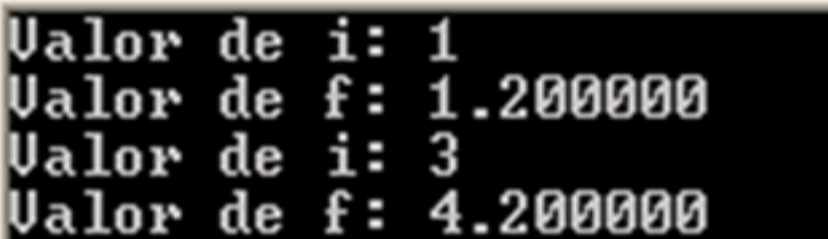
```
    (*p_minha_estrutura).f = 4.2;
```

```
    printf("Valor de i: %d\n", (*p_minha_estrutura).i);
```

```
    printf("Valor de f: %f\n", (*p_minha_estrutura).f);
```

```
    system("pause");
```

```
}
```



```
Valor de i: 1
Valor de f: 1.200000
Valor de i: 3
Valor de f: 4.200000
```

# PONTEIRO COMO ARGUMENTO EM UMA FUNÇÃO

- **Passar um vetor para uma função** consiste:
  - em passar o endereço da **primeira posição do vetor**, ou seja, passar o **endereço inicial do vetor**.
- Os elementos do vetor **não são copiados** para a função, o argumento copiado é apenas o **endereço** do **primeiro elemento** do array.
- **Exemplo:**

*Se passarmos para uma função um vetor de int, devemos ter um parâmetro do tipo int \*, capaz de armazenar endereços de inteiros. Logo, se passarmos um valor de endereço, a função chamada deve ter um parâmetro do tipo ponteiro para armazenar este valor.*

- Fazer a troca dos valores de duas variáveis.
- Chamada por valor, pois foi passado diretamente os valores das variáveis para uma função. Ops! Resultado!

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

//funcao que realiza a troca dos valores entre duas variaveis
void troca(int i, int j)
{
    int temp;
    temp = i;
    i = j;
    j = temp;
}

int main()
{
    int a, b;
    a = 5;
    b = 10;
    printf ("Valores de origem: %d %d\n", a, b);
    troca(a, b);
    printf ("Valores trocados: %d %d\n", a, b);

    system("pause");

    return 0;
}
```

```
Valores de origem: 5 10
Valores trocados: 5 10
Pressione qualquer tecla para continuar. . .
```

- Trocar o conteúdo de 2 variáveis, utilizando ponteiros nos argumentos da função (chamada por referência).

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

//funcao que realiza a troca dos valores entre duas variaveis
void troca(int *i, int *j)
{
    int temp;
    temp = *i;
    *i = *j;
    *j = temp;
}

int main()
{
    int a, b;
    a = 5;
    b = 10;
    printf ("Valores de origem: %d %d\n", a, b);
    troca(&a, &b);
    printf ("Valores trocados: %d %d\n", a, b);

    system("pause");

    return 0;
}
```

Valores de origem: 5 10  
Valores trocados: 10 5

- **Ponteiro como argumento em uma função**
- **Ex.: passar para a função o endereço do primeiro elemento do vetor (e não os elementos propriamente ditos), para alterar os valores dos elementos do vetor dentro da função.**

```
/* Incrementa elementos de um vetor */  
  
#include <stdio.h>  
  
void incr_vetor ( int n, int *v )  
{  
    int i;  
    for (i = 0; i < n; i++)  
        v[i]++;  
}  
  
int main ( void )  
{  
    int a[ ] = {1, 3, 5};  
    incr_vetor(3, a);  
    printf("%d %d %d \n", a[0], a[1], a[2]);  
    return 0;  
}
```

**Saída do programa:**

**2 4 6**

**Os elementos do vetor serão incrementados dentro da função.**

# PONTEIRO PARA FUNÇÃO

- Um **ponteiro para função contém o endereço da função** da memória.
- Semelhantemente a uma matriz de ponteiro, um **nome de função** é o **endereço** na **memória** do **começo** do **código** que executa a tarefa da função.
- O uso mais comum de ponteiros para funções é permitir que uma **função** possa ser **passada para uma outra função**, para isso um parâmetro ponteiro deverá ser indicado e com isso quando a **função for executada um valor de argumento é passado para outra função**.
- Funções que devolvem ponteiros funcionam da mesma forma que os outros tipos de funções.
- Uma função que retorna ponteiro deve **declarar qual tipo de ponteiro está retornando**.

# PONTEIRO PARA FUNÇÃO

- Ponteiros de função podem ser:
  - atribuídos a outros ponteiros;
  - passados como argumentos;
  - retornados por funções; e,
  - armazenados em matrizes.

- **Sintaxe:**

```
tipo_retorno (*nome_do_ponteiro)(lista de argumentos)
```

- **<tipo> não** pode ser *void*, pois:
  - a função deve devolver algum valor;
  - ponteiro deve apontar para algum tipo de dado.


- A função soma retorna a soma dos 2 inteiros a ela fornecidos. Há uma chamada indireta, por meio da função operacao, através de um ponteiro. A main passa a função soma como argumento para operacao, e a função operacao chama essa função que lhe foi dada como argumento.

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

//funcao para somar dois valores
int soma(int a, int b)
{
    return (a + b);
}

//funcao chama a funcao soma passada por referencia como argumento
int operacao(int x, int y, int (*func)(int,int))
{
    int g;
    g = (*func)(x, y);
    return (g);
}

int main ()
{
    int m;
    m = operacao(7, 5, soma);
    printf("Soma de 7+5 = %d\n", m);
    system("pause");
    return 0;
}
```





# PONTEIRO PARA FUNÇÃO

- *O terceiro argumento da função operacao é um ponteiro para uma função.*
- *Nesse caso, ele foi declarado como um ponteiro para uma função que toma dois inteiros como argumentos e retorna outro inteiro.*
- *O \* indica que estamos declarando um ponteiro, e não uma função.*
- *Os parênteses em torno de \*func são essenciais, pois sem eles o compilador entenderia o argumento como uma função que retorna um ponteiro para um inteiro.*

# PONTEIRO PARA FUNÇÃO

- Para **chamar a função apontada pelo ponteiro**, há duas sintaxes. A sintaxe original é:

```
(*nome_do_ponteiro)(argumentos);
```

- Se *ptr* é um ponteiro para uma função, faz bastante sentido que a função em si seja chamada por *\*ptr*.
- Há uma sintaxe mais moderna permite que ponteiros para funções sejam chamados exatamente da mesma maneira que funções:  

```
nome_do_ponteiro(argumentos);
```
- Para inicializar um ponteiro para função, não precisamos usar o operador de endereço &. Por isso, quando chamamos a função *operacao*, não precisamos escrever *&soma*.

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

int soma(int a, int b)
{
    return (a+b);
}

int subtracao(int a, int b)
{
    return (a-b);
}

//declara/cria um ponteiro para a funcao subtracao chamado menos
int (*menos)(int, int) = subtracao;

int operacao(int x, int y, int (*func)(int,int))
{
    int g;
    g = func(x, y); //sintaxe moderna para a chamada de ponteiro de funcao
    return (g);
}

int main()
{
    int m, n;
    m = operacao(7, 5, soma);
    //referimo-nos à função de subtração através do ponteiro menos
    n = operacao(20, m, menos);
    printf("Soma 7+5= %d\n", m);
    printf("Subtrai 20-m= %d\n", n);

    system("pause");

    return 0;
}
```

```
Soma 7+5= 12
Subtrai 20-m= 8
Pressione qualquer tecla para continuar. . .
```

## DESAFIOS...



Para todas os programas, abaixo, utilize ponteiros para acessar e manipular indiretamente os valores contidos nos mesmos. O programa deve:

1. Faça um programa para ler 20 números inteiros, calcular a média dos mesmos e exibir todos os números que estiverem acima da média. Considere a média como sendo 7.0.
2. Dado um conjunto de 20 valores reais armazenados em um vetor, faça um algoritmo que:
  - a) Imprima os valores que não são negativos.
  - b) Calcule e imprima a média dos valores negativos.

## DESAFIOS...



3. Crie uma matriz quadrada e a inicialize de forma que seja atribuído o valor 2 quando os índices forem iguais e -3 quando os índices forem diferentes. Calcule e imprima na tela apenas o somatório da diagonal principal.
4. Escreva um programa que tenha uma matriz de 12 elementos quaisquer informados pelo usuário e imprima quantos elementos são pares e quantos são ímpares, bem como a soma dos números pares e a soma dos números ímpares.