

# ESTRUTURA DE DADOS

## AULA 05 – PONTEIROS (APONTADORES)

PARTE II





# PONTEIRO - RELEMBRANDO

- **Para manipular há dois operadores especiais:**
  - o de **endereço**: **&**
  - o de **conteúdo**: **\***
- **Declaração de uma variável como ponteiro:**  
*int \*x; // \* precedendo o nome do ponteiro*
- **Acessar o conteúdo da variável para o qual o ponteiro aponta:**  
*y = \*x; // o \* retorna o valor apontado pelo ponteiro*
- **Inicializar um ponteiro com um endereço:**  
*px = &x; //px aponta para x, pois contem o endereço de x*  
*px=NULL; //px não aponta para nenhum endereço*
- **Igualar ponteiros:**  
*px=py; //conterão o mesmo endereço e apontarão para o mesmo lugar*
- **Incrementar/decrementar ponteiros:**  
*px++; //incrementa o endereço referenciado pelo ponteiro de acordo com o tipo de dado para o qual o ponteiro aponta.*

# PONTEIRO - RELEMBRANDO

- **Trabalhar com ponteiros, envolve basicamente:**
  - **conhecer endereço de uma variável;**
  - **conhecer/acessar o conteúdo de um endereço.**
- **Há vários tipos de ponteiro:**
  - **ponteiros para caracteres;**
  - **ponteiros para inteiros;**
  - **ponteiros para ponteiros para inteiros;**
  - **ponteiros para vetores;**
  - **ponteiros para estruturas;**
  - **Ponteiros para funções.**

# PONTEIRO GENÉRICO

- é um ponteiro que pode **apontar** para **qualquer tipo de dado**, inclusive para outro ponteiro.
- é aquele que pode **apontar** para **todos** os **tipos** de dados existentes ou que ainda **serão criados**;
- **Sintaxe:**  
***void \*<nome\_ponteiro>;***

```
1 #include <stdio.h>
2 #include <stdlib.h>
3
4 int main () {
5     void *pp;
6     int *p1, p2 = 10;
7     p1 = &p2; //recebe o endereço de um inteiro
8     pp = &p2;
9     printf("Endereco em pp: %p \n", pp);
10    //recebe o endereço de um ponteiro para inteiro
11    pp = p1;
12    printf("Endereco em pp: %p \n", pp);
13    //recebe o endereço guardado em p1 (endereço de p2)
14    printf("Endereco em pp: %p \n", pp);
15    system("pause");
16    return 0;
17 }
```

# PONTEIRO GENÉRICO

- Para **acessar** o **conteúdo** **referenciado** por um ponteiro **genérico** é necessário utilizar o **operador** de **type cast** antes do **identificador** do **ponteiro**, e isto deve ocorrer **antes** em uma **operação** que esteja **manipulando** este **conteúdo**, do contrário ocorrerá **erro**.
- É necessário **converter** o ponteiro **genérico** para o **tipo de dado** com o qual se **deseja trabalhar antes de acessar** o seu **conteúdo**.
- Type Cast:
  - é chamado **modelador** de tipos;
  - é uma forma **explícita** de **conversão** de tipo, onde o tipo a ser **convertido** é explicitamente definido dentro de um programa.
  - É diferente da conversão implícita, que ocorre quando tentamos atribuir um número real para uma variável inteira.

# TYPE CAST

- Sintaxe:  
*(nome do tipo) expressão*
- Um modelador de tipo é **definido** pelo **próprio nome do tipo** entre **parênteses** e é **colocado à frente** de uma **expressão** e tem como objetivo **forçar** o **resultado da expressão a ser de um tipo especificado**.

```
1 #include <stdio.h>
2 #include <stdlib.h>
3 int main() {
4     float x,y,f = 65.5;
5     x = f/10.0;
6     y = (int) (f/10.0);
7     printf('x = %f\n',x);
8     printf('y = %f\n',y);
9     system('pause');
10    return 0;
11 }
```

Saída

x = 6.550000

y = 6.000000

○ Exemplo: manipulando o ponteiro genérico + *type cast*

```
1 #include <stdio.h>
2 #include <stdlib.h>
3
4 int main () {
5     void *pp;
6     int p2 = 10;
7     // ponteiro genérico recebe o endereço de um
        inteiro
8     pp = &p2;
9     //enta acessar o conteúdo do ponteiro genérico
10    printf( ' 'Conteudo: %d\n' ',*pp); //ERRO
11    //converte o ponteiro genérico pp para (int *)
        antes de acessar seu conteúdo.
12    printf( ' 'Conteudo: %d\n' ',*(int*)pp); //
        CORRETO
13    system( ' 'pause ' ');
14    return 0;
15 }
```

# PONTEIRO GENÉRICO: ARITMÉTICA

- Como o ponteiro **genérico não** possui **tipo definido**, deve-se ficar **atento** às **operações matemáticas** que serão **realizadas** sobre os **endereços contidos** nos **ponteiros**.
- As operações aritméticas **não funcionam em ponteiros genéricos** da **mesma forma** como em ponteiros de tipos definidos, pois com o ponteiro **genérico** são sempre **realizadas** com base em uma **unidade de memória (1 byte)**.
- As operações de **adição** e **subtração** com o ponteiro genérico **adicionados/subtraídos 1byte** por incremento/decremento, pois este é o tamanho de uma unidade de memória.
- Ex.: se o endereço guardado for, por exemplo, de um inteiro, o incremento de uma posição no ponteiro genérico (1 byte) não irá levar ao próximo inteiro (4 bytes).



## ○ Exemplo: operações aritmética com o ponteiro genérico

```
1 #include <stdio.h>
2 #include <stdlib.h>
3 int main() {
4     void *p = 0x5DC;
5     printf('p = Hexadecimal: %p Decimal: %d \n',
6           p, p);
7     p++; // Incrementa p em uma posição
8     printf('p = Hexadecimal: %p Decimal: %d \n',
9           p, p);
10    // Incrementa p em 15 posições
11    p = p + 15;
12    printf('p = Hexadecimal: %p Decimal: %d \n',
13          p, p);
14    // Decrementa p em 2 posições
15    p = p - 2;
16    printf('p = Hexadecimal: %p Decimal: %d \n',
17          p, p);
18    system('pause');
19    return 0;
20 }
```

Saída

p = Hexadecimal: 000005DC Decimal: 1500

p = Hexadecimal: 000005DD Decimal: 1501

p = Hexadecimal: 000005EC Decimal: 1516

p = Hexadecimal: 000005EA Decimal: 1514

# PONTEIROS E VETOR

- O **nome** do **vetor/matriz** nada mais é do que um **ponteiro** que **aponta** para o **primeiro elemento do array**.
- O nome do *array*, sem índice, está associado ao endereço para o início do *array* na memória. Por isso, qualquer operação que possa ser feita com índices de um vetor pode também ser realizada com **ponteiros e aritmética de ponteiros**. Ex:
  - `int v[10];`
  - a variável *v*, que representa o vetor, é uma **constante** que **armazena o endereço inicial do vetor**, isto é, ***v*, sem indexação, aponta para o primeiro elemento do vetor**.
  - Ex.: `scanf("%f", &v[i]);`
  - Se *v[i]* representa o (i+1)-ésimo elemento do vetor, *&v[i]* representa o endereço de memória onde esse elemento está armazenado.

# PONTEIROS E VETOR

- Escrever **&v[i]** é equivalente a escrever **(v+i)**. De maneira análoga, escrever **v[i]** é equivalente a escrever **\*(v+i)**.
- O uso da aritmética de ponteiros pode ser **aplicada**, pois os elementos dos **vetores** são armazenados de forma **contígua** na memória.
- Há **duas** formas de **indexar** os **elementos** de um *array*:
- usando o operador de indexação, ou seja, o índice do arranjo. Ex.:

`v[4]`      *//obtemos o conteúdo do vetor*

- usando a aritmética de endereços com os ponteiros.
- Ex.:

`*(v+4)` *//v é ponteiro; retorna o conteúdo referenciado pelo ponteiro.*

# PONTEIROS PARA VETOR

- Considerando as declarações abaixo:

```
int x, a[10];  
int *pa;
```

- Se for feito:

```
pa=&a[0];  
pa=a;           //passa o endereço inicial do vetor "a" para o ponteiro pa  
x=*pa;          //(passa o conteúdo de a[0] para x
```

- Logo: Se ***pa*** aponta para um elemento particular de um vetor “a”, então por definição ***pa+1*** aponta para o próximo elemento.
- Em geral, ***pa-i*** aponta para *i* elementos antes de ***pa*** e ***pa+i*** para *i* elementos depois.
- Se ***pa*** aponta para ***a[0]***, então:  
    ***\*(pa+1)*** aponta para o elemento de ***a[1]***  
    ***pa+i*** é o endereço de ***a[i]*** e ***\*(pa+i)*** é o conteúdo.

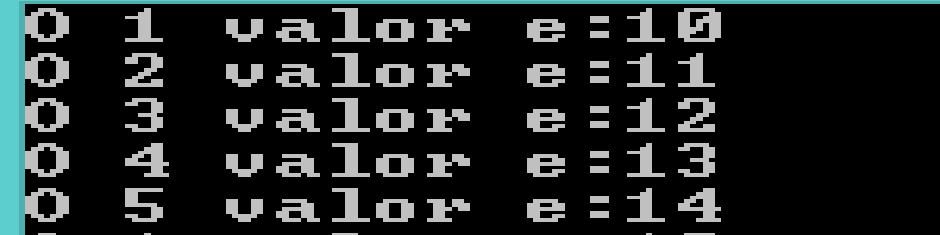
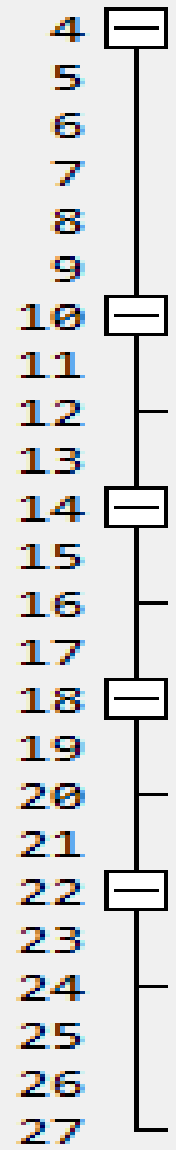
✓ Ex:

```
1 #include <stdio.h>
2 #include <stdlib.h>
3 int main () {
4     int vet[5] =
        {1,2,3,4,5};
5     int *p = vet;
6     int i;
7     for (i = 0; i < 5; i++)
8         printf("%d\n", p[i]);
9     system("pause");
10    return 0;
11 }
```

```
1 #include <stdio.h>
2 #include <stdlib.h>
3 int main () {
4     int vet[5] =
        {1,2,3,4,5};
5     int *p = vet;
6     int i;
7     for (i = 0; i < 5; i++)
8         printf("%d\n", *(p+
            i));
9     system("pause");
10    return 0;
11 }
```

## Exemplos: Ponteiro e Vetores

```
1  #include <stdio.h>
2  #include <stdlib.h>
3
4  main(){
5
6      int v[5]={10,11,12,13,14};
7      int *pv=v;
8
9      //Quatro formas de acessar vetor
10     for (int i=0;i<5;i++){
11         printf("0 %d valor e:%d\n",i+1, v[i]);
12     }
13
14     for (int i=0;i<5;i++){
15         printf("0 %d valor e:%d\n",i+1, *(v+i));
16     }
17
18     for (int i=0;i<5;i++){
19         printf("0 %d valor e:%d\n",i+1, pv[i]);
20     }
21
22     for (int i=0;i<5;i++){
23         printf("0 %d valor e:%d\n",i+1, *(pv++));
24     }
25
26     system("PAUSE");
27 }
```



Index	valor	e
0	10	10
1	11	11
2	12	12
3	13	13
4	14	14



- Ponteiros **permitem percorrer** as várias **dimensões** de um **array multidimensional** como se existisse **apenas** uma **dimensão**. Ex:

### Usando Array

```
1 #include <stdio.h>
2
3 int main () {
4     int mat[2][2] =
5         {{1,2},{3,4}};
6     int i,j;
7     for(i=0;i<2;i++)
8         for(j=0;j<2;j++)
9             printf("%d\n",mat[i][j]);
10
11     return 0;
12 }
```

### Usando Ponteiro

```
1 #include <stdio.h>
2
3 int main () {
4     int mat[2][2] =
5         {{1,2},{3,4}};
6     int * p = &mat[0][0];
7     int i;
8     for(i=0;i<4;i++)
9         printf("%d\n",*(p+i));
10
11     return 0;
12 }
```

# PRIMEIROS PASSOS

Faça um programa que utiliza ponteiros para acessar e manipular indiretamente os valores contidos nos mesmos. O programa deve:

1º :

ler um vetor de 100 números reais fornecidos pelo usuário ;

2º :

ler um outro valor real, fornecido pelo usuário;

3º :

verifique se este outro valor real existe ou não no vetor de 100 elementos. Caso exista, exiba a quantidade de vezes que este número se repete. Caso contrário, exiba uma mensagem ao usuário.

4º :

imprimir na tela todos os números maiores que o valor real fornecido pelo usuário.

