

# **Processamento de Dados II**

## **Prof. Max Davi**

# Ponteiros

## Memória do PC

Endereço



0	
1	
2	
3	
4	
5	
6	
7	
8	
9	
10	
11	
12	
13	
14	
15	
16	

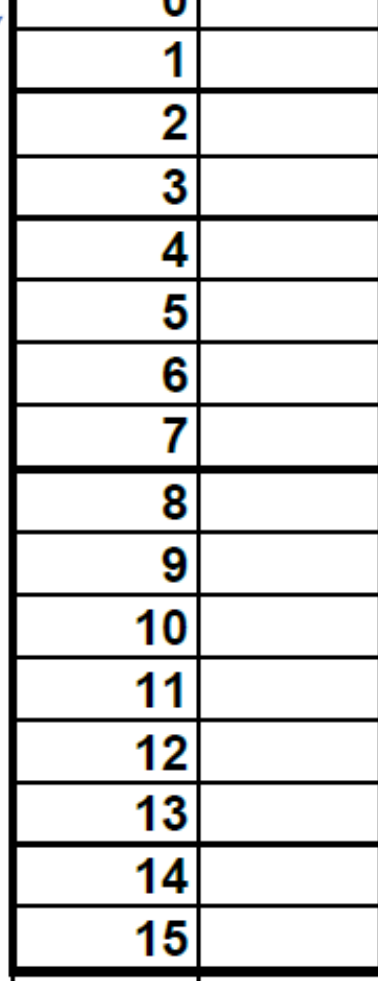
**1 Byte = 8 bits**

**1024 Bytes = 1 KiloBytes**

**1024 KiloBytes = 1 MegaByte**

## Memória do PC

Endereço



0	
1	
2	
3	
4	
5	
6	
7	
8	
9	
10	
11	
12	
13	
14	
15	
16	

1 Byte = 8 bits


1024 Bytes = 1 KiloBytes

1024 KiloBytes = 1 MegaByte

int x;

## Memória do PC

Endereço



0	
1	
2	
3	
4	
5	
6	
7	
8	
9	
10	
11	
12	10
13	
14	
15	
16	

1 Byte = 8 bits

1024 Bytes = 1 KiloBytes

1024 KiloBytes = 1 MegaByte

int x;

x=10

# Conceito de ponteiros

**Ponteiro é uma variável que contém um endereço de memória. Esse endereço pode ser a posição de uma outra variável na memória. Se uma variável contém o endereço de uma outra, então a primeira variável é dita para apontar para a segunda.**

Endereço na  
memória

Variável na  
memória

1000	1003
1001	
1002	
1003	
1004	
1005	
1006	

•  
•  
•

Memória

# Como funcionam os ponteiros

Ponteiros guardam endereços de memória. Por exemplo: quando você anota o endereço de um colega você está criando um ponteiro. O ponteiro é este seu pedaço de papel. Ele tem anotado um endereço. Qual é o sentido disto? Simples. Quando você anota o endereço de um colega, depois você vai usar este endereço para achá-lo. O C funciona assim. Você anota o endereço de algo numa variável ponteiro para depois usar.



# Por que usar ponteiro?

**Ponteiros são muito úteis quando uma variável tem que ser acessada em diferentes partes de um programa.**

**Neste caso, o código pode ter vários ponteiros espalhados por diversas partes do programa, “apontando” para a variável que contém o dado desejado.**

**Caso este dado seja alterado, não há problema algum, pois todas as partes do programa tem um ponteiro que aponta para o endereço onde reside o dado atualizado.**

# Por que usar ponteiro?

**Existem várias situações onde ponteiros são úteis, por exemplo:**

- ▶ **Alocação dinâmica de memória**
- ▶ **Manipulação de arrays**
- ▶ **Referência para listas, pilhas, árvores e grafos**

# Tipo de ponteiro

**No C quando declaramos ponteiros nós informamos ao compilador para que tipo de variável vamos apontá-lo. Um ponteiro int aponta para um inteiro, isto é, guarda o endereço de um inteiro.**

# Declaração de ponteiro

Para declarar um ponteiro temos a seguinte forma geral:

*tipo\_do\_ponteiro \*nome\_da\_variável;*

# Operadores de ponteiros

Existem dois operadores especiais para ponteiros:

- ▶ **&:** Operador unário que devolve o endereço na memória de seu operando.

**Ex: `m = &count;`**

- ▶ No exemplo acima, coloca-se em `m` o endereço que contem a variável `count`. O operador `&` pode ser imaginado como retornando o “endereço de”. Assumindo então que `count` usa a posição de memória “2000” para armazenar seu valor, mas o valor de `count` é “100”. Então, após a atribuição anterior, `m` terá o valor de “2000”.

# Operadores de ponteiros

- ▶ **\*:** Operador unário que devolve valor da variável localizada no endereço que o segue. Por exemplo, se **m** contém o endereço da variável **count** como no exemplo anterior,

**Ex:  $q = *m;$**

- ▶ No exemplo acima, coloca-se o valor de **count** em **q**. Portanto, **q** terá o valor de 100 porque 100 estava armazenado na posição 2000, que é o endereço que estava armazenado em **m**. O operador **\*** pode ser imaginado como “no endereço”. Nesse caso, o comando anterior significa “**q** recebe o valor que está no endereço **m**”.

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

```
#include <conio.h>
```

```
int main(void)
```

```
{
```

```
    int valor = 27; //variável que será apontada pelo ponteiro
```

```
    int *ptr; //declaração de variável ponteiro
```

```
    ptr = &valor; //atrib. endereço da variável valor ao ponteiro
```

```
    printf("Utilizando ponteiros\n\n");
```

```
    printf ("Conteúdo da variavel valor: %d\n", valor);
```

```
    printf ("Endereço da variavel valor: %x \n", &valor);
```

```
    printf ("Conteúdo da variavel ponteiro ptr: %x", ptr);
```

```
    return(0);
```

```
}
```

**Obs: Foi utilizado %x para exibir o endereço e o conteúdo do ponteiro ptr, pois trata-se de um valor hexadecimal por ser endereço de memória.**

# Operadores de ponteiros

`int *m; //Declaração do ponteiro m`

`&m //Endereço que está sendo acessado pelo ponteiro a`

`*m //Valor da variável para a qual o ponteiro aponta`

**Exemplo:**

`int count, q;`

`int *m;`

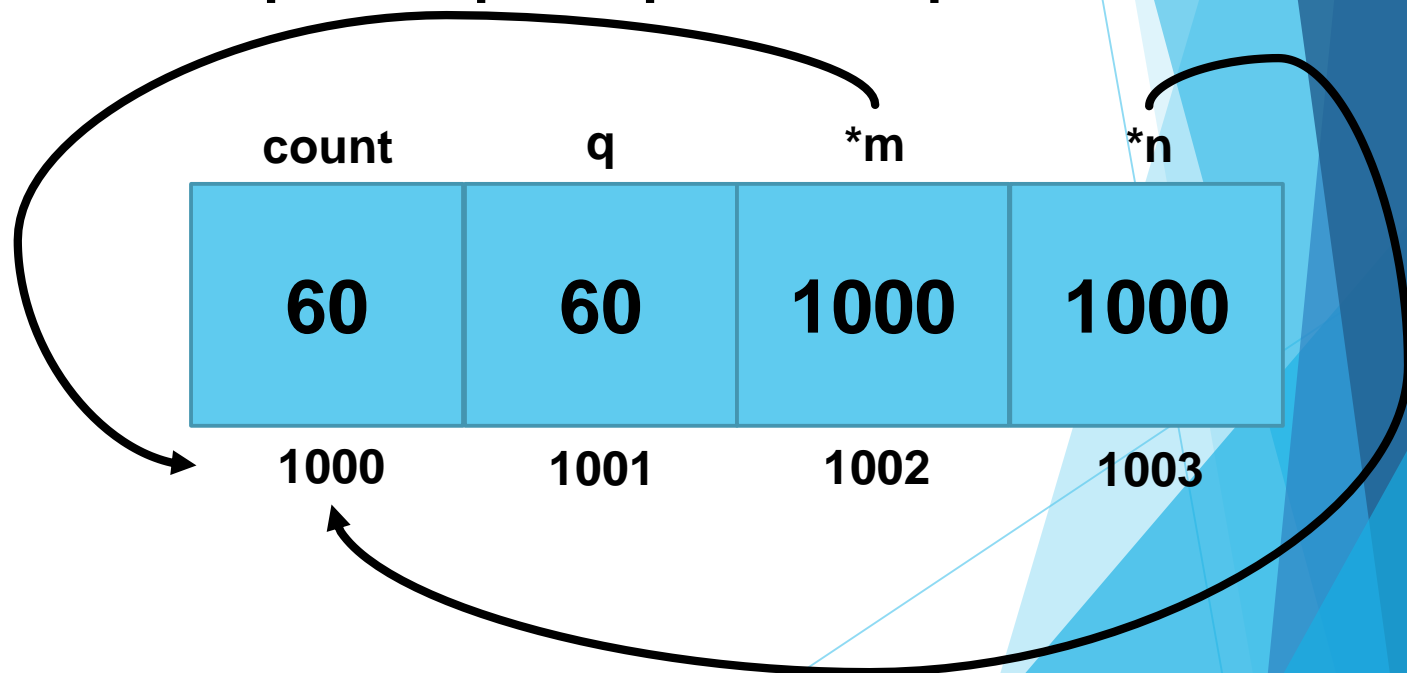
`int *n;`

`m = &count;`

`count = 60;`

`q = *m;`

`n = m;`





# Inicialização do ponteiro

**O ponteiro deve ser inicializado (apontado para algum lugar conhecido) antes de ser utilizado.**

**Para atribuir um valor a um ponteiro recém-criado poderíamos igualá-lo a um valor de memória. Mas, como saber a posição na memória de uma variável do nosso programa?**

**Podemos deixar o compilador fazer esse trabalho. Para saber o endereço de uma variável basta usar o operador &.**

# Inicialização do ponteiro

**Exemplo:**

```
int count=10;  
int *ponteiro;  
ponteiro=&count;
```

**Criamos um inteiro count com o valor 10 e um apontador para um inteiro ponteiro. A expressão &count nos dá o endereço de count, o qual armazenamos em ponteiro.**

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

```
int main ()
```

```
{
```

```
int num,valor;
```

```
int *p;
```

```
num=55;
```

```
p=&num; /* Pega o endereço de num */
```

```
valor=*p; /*Valor e igualado a num indiretamente*/
```

```
printf ("\n\n%d\n",valor);
```

```
printf ("Endereço onde o ponteiro aponta:%p\n",p);
```

```
printf ("Valor da variavel apontada: %d\n",*p);
```

```
return(0);
```

```
}
```

Obs: O %p usado na função printf() indica deve imprimir o endereço.

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

```
int main ()
```

```
{
```

```
int num,*p;
```

```
num=55;
```

```
p=&num; /* Pega o endereco de num */
```

```
printf ("\nValor inicial: %d\n",num);
```

```
*p=*p+100; /* Muda o valor de num indiretamente*/
```

```
printf ("\nValor final: %d\n",num);
```

```
return(0);
```

```
}
```

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

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

```
int main(void)
```

```
{
```

```
    int x, *ptrx, **pptrx;
```

```
    //inicializando a variavel com zero
```

```
    x = 0;
```

```
    printf("\nvalor de x = %d\n", x);
```

```
    printf("Endereco de x: %x\n\n",&x);
```

```
    //Atribuindo os enderecos para os ponteiros
```

```
    ptrx = &x; // ptrx aponta para x
```

```
    pptrx = &ptrx; // pptrx aponta para ptrx
```

```
*ptrx = *ptrx + 10;
printf("\nvalor de x = %d", x);
printf("\nEndereco apontado por ptrx: %x\n",ptrx);
printf("Valor da variavel X que esta sendo apontada por
ptrx: %d\n",*ptrx);
printf("Endereco de memoria da variavel ptrx %x\n",&ptrx);

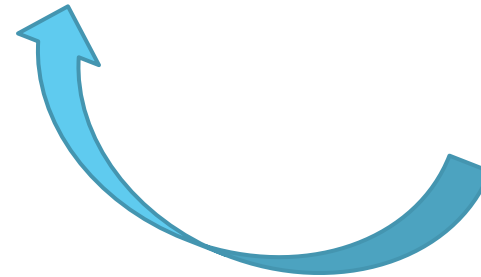
**pptrx = **pptrx + 10;
printf("\n\nvalor de x = %d", x);
printf("\nEndereco apontado por **pptrx: %x",pptrx);
printf("\nValor da variavel para a qual pptrx faz referencia:
%d",**pptrx);
printf("\nEndereco de memoria da variavel **pptrx
%x\n\n",&pptrx);
return 0;
}
```

```
x=0;  
ptrx=&x;  
pptrx=&ptrx;
```

x
0
62FE4C

ptrx
62FE4C
62FE40

pptrx
62FE40
62FE38



# Exercício

**Escreva um programa que contenha duas variáveis inteiras. Compare seus endereços e exiba o maior endereço.**



```
#include <stdio.h>
#include <stdlib.h>
int main (void){
int a = 10, b = 100;
int *EndA = &a , *EndB = &b;
printf("\tA = %d    B = %d\n\n",a, b);
printf("Endereco de A = %x\nEndereco de B = %x\n",EndA,
EndB);
    if(EndA > EndB){
        printf("\n\nEndereco %x de A e maior: \n",EndA);
    }
    else{
        printf("\n\nEndereco %x de B e maior: \n",EndB);
    }
}
```

# Exercício

**Escreva um programa que contenha 2 variáveis inteiras. Leia essas variáveis do teclado. Em seguida, compare seus endereços e exiba o maior endereço.**

```
#include <stdio.h>
#include <stdlib.h>
int main (void){
int a, b;
int *EndA = &a, *EndB = &b;
printf("Digite o valor de a: ");
scanf("%d",&a);
printf("\nDigite o valor b: ");
scanf("%d",&b);
printf("\nValor de a = %d\nValor de b = %d\n\n",a, b);
printf("Endereco de a = %x\nEndereco de b = %x\n\n",EndA, EndB);
if(EndA > EndB){
printf("\n\nEndereco %x de a e maior:",EndA);
}
else{
printf("\n\nEndereco %x de b e maior:", EndB);
}
return 0;
}
```

# Operações com ponteiros

**Quando incrementamos um ponteiro ele passa a apontar para o próximo valor do mesmo tipo para o qual o ponteiro aponta. Isto é, se temos um ponteiro para um inteiro e o incrementamos ele passa a apontar para o próximo inteiro. Esta é mais uma razão pela qual o compilador precisa saber o tipo de um ponteiro: se você incrementa um ponteiro `char*` ele anda 1 Byte na memória e se você incrementa um ponteiro `double*` ele anda 8 Bytes na memória. O decremento funciona semelhantemente.**

# Operações com ponteiros

**Igualar dois ponteiros (p1 e p2):**

- ▶ **p1=p2 (P1 aponta mesmo local de P2)**

**Igualar o conteúdo (p1 e p2):**

- ▶ **\*p1=\*p2 (P1 terá mesmo conteúdo de P2)**

**Incremento e decremento:**

- ▶ **p1++ ou p1-- (O ponteiro ele passa a apontar para o próximo valor do mesmo tipo para o qual o ponteiro aponta ou o valor anterior no caso de decremento)**

# Operações com ponteiros

**Incremento do conteúdo da variável:**

▶ **`(*p)++;`**

**Incremento do ponteiro em 15:**

▶ **`p=p+15` ou `p+=15;`**

**Incremento do conteúdo da variável em 15:**

▶ **`*p1=*p1+15;`**

# Comparação de ponteiros

Podemos saber se dois ponteiros são iguais ou diferentes:

- ▶ **== (Igual)**
- ▶ **!= (Diferente)**

E se apontam para uma posição mais alta na memória:

- ▶ **>, <, >= e <=**

# Operações não efetuadas por ponteiros

- ▶ dividir ou multiplicar ponteiros
- ▶ adicionar dois ponteiros
- ▶ adicionar ou subtrair **floats** ou **doubles** de ponteiros



# Atividade Final

**a) Escreva um programa que declare uma variável do tipo inteiro, uma real e um char, e que tenhamos ponteiros para inteiro, real, e char. Associe as variáveis aos ponteiros.**

**O programa deve permitir que se modifique os valores de cada variável utilizando ponteiros. Deve-se apresentar na tela os valores das variáveis antes e após a modificação.**

# Atividade Final

**b) Desenvolva um programa em C que declare duas variáveis do tipo float e duas do tipo ponteiro de float apontando para essas variáveis. Utilizando ponteiros, o programa deve ler dois números para essas variáveis e os imprimir, realizando as quatro operações básicas de matemática com esses números.**

# Atividade Final

**c) Desenvolva um programa em C que declare três variáveis do tipo inteiro e três do tipo ponteiro de inteiro apontando para essas variáveis. Utilizando ponteiros, leia três números e os imprima em ordem crescente. O programa deve apresentar também o endereço de memória desses números.**

# Referências Bibliográficas

- Deitel H and Deitel P. - C: Como Programar, 6 edição, Pearson;
- Schildt H. - C Completo e Total – Makron Books;
- Ana Fernanda Gomes Ascencio e Edilene Aparecida Veneruchi de Campos - Fundamentos da Programação de Computadores: Algoritmos, Pascal, C, C++ e Java.