



---

# **Programação II**

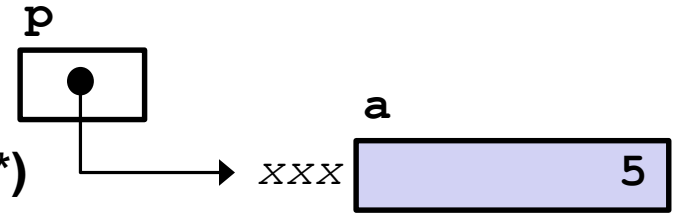
## **Ponteiros**

**Bruno Feijó**  
**Dept. de Informática, PUC-Rio**

# Ponteiro

- Ponteiro (pointer) é uma variável que armazena o endereço de uma variável.

Sempre inicialize ponteiros ! Ponteiros não inicializados são “*wild pointers*” (ponteiros selvagens); muito perigosos !
- Declaramos um ponteiro usando o caractere (\*)
  - `int * p;`                      `float * p;`                      `char * s;`
  - Na declaração, evite juntar \* ao nome da variável (e.g. evite `int *p`), pois a propriedade de ser um ponteiro é uma propriedade do tipo e não da variável.
- Trabalhamos com ponteiros usando os operadores \* e &
  - O operador unário \* (também chamado de operador *indirection* ou *dereferencing* - no sentido de ser indireto, de ser um derivativo de referenciar) e que pode ser lido como “**conteúdo de**”: quando aplicado a um ponteiro ele acessa o conteúdo da variável que ele aponta.
  - O operador unário & (leia-se “**endereço de**”): quando aplicado a uma variável resulta no endereço de memória reservado para esta variável.



```
int a = 5;
int * p;
p = &a;
int b = *p; // b = 5
```

# Exemplo - ponteiro

```
int a;  
int * p;
```

p	-	108
a	-	104

```
/* a recebe o valor 5 */  
a = 5;
```

```
/* p recebe o endereço de a  
ou seja, p aponta para a */  
p = &a;
```

```
/* posição de memória apontada por p  
recebe 6 */  
*p = 6;
```

```
/* c recebe o valor armazenado  
na posição de memória apontada por p */  
c = *p;
```

c	-	112
p	-	108
a	5	104

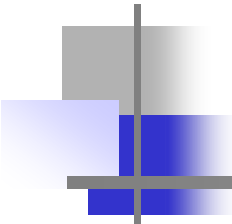
c	-	112
p	104	108
a	5	104

c	-	112
p	104	108
a	6	104

c	6	112
p	104	108
a	6	104

# Incrementando/Decrementando Ponteiros

```
int a = 5;
int * p;
p = &a;
int b = *p; // b = 5
int x = *p + 1 // x = 6, mas a continua 5
*p += 1 // *p = *p + 1 = 6, a passa a ser 6 tambem
++*p; // *p = 7 = a
(*p)++; // *p = 8 = a
*p++; // significa *(p++), i.e. p = p+1 contem
// o proximo endereco e a fica inalterado
// e igual a 8
```



## Exemplo - ponteiro

```
int main(void)
{
    int a;
    int * p;
    p = &a;
    *p = 2;
    printf(" %d ", a);
    return 0;
}
```

imprime o valor 2

## Exemplo - ponteiro

```
int main(void)
{
    int a, b, * p;
    a = 2;
    *p = 3;
    b = a + (*p);
    printf(" %d ", b);
    return 0;
}
```

– erro na atribuição **\*p = 3**

- utiliza a memória apontada por p para armazenar o valor 3, sem que p tivesse sido inicializada, logo
- armazena 3 num espaço de memória desconhecido

## Exemplo - ponteiro

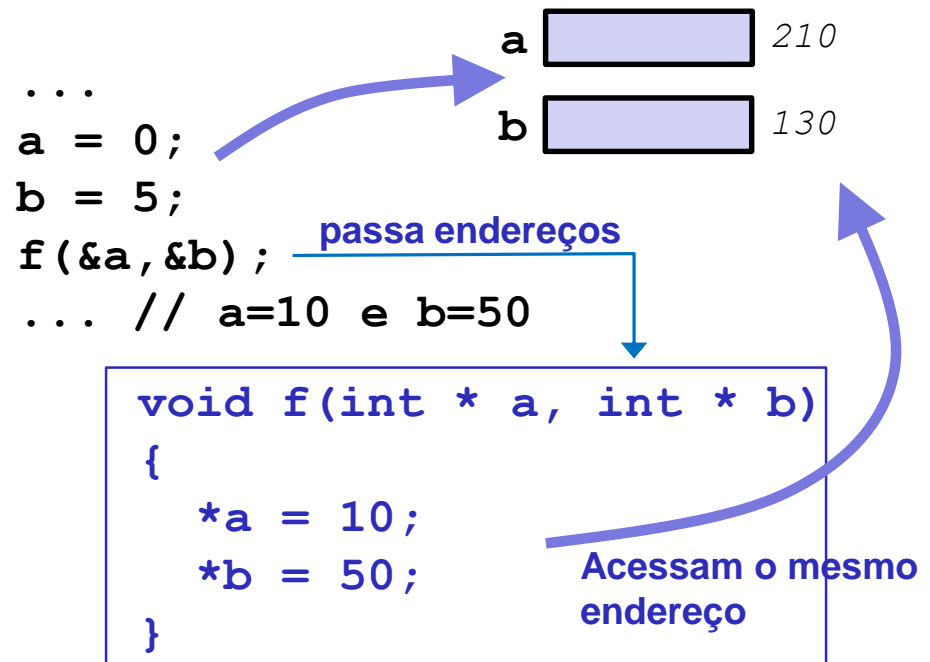
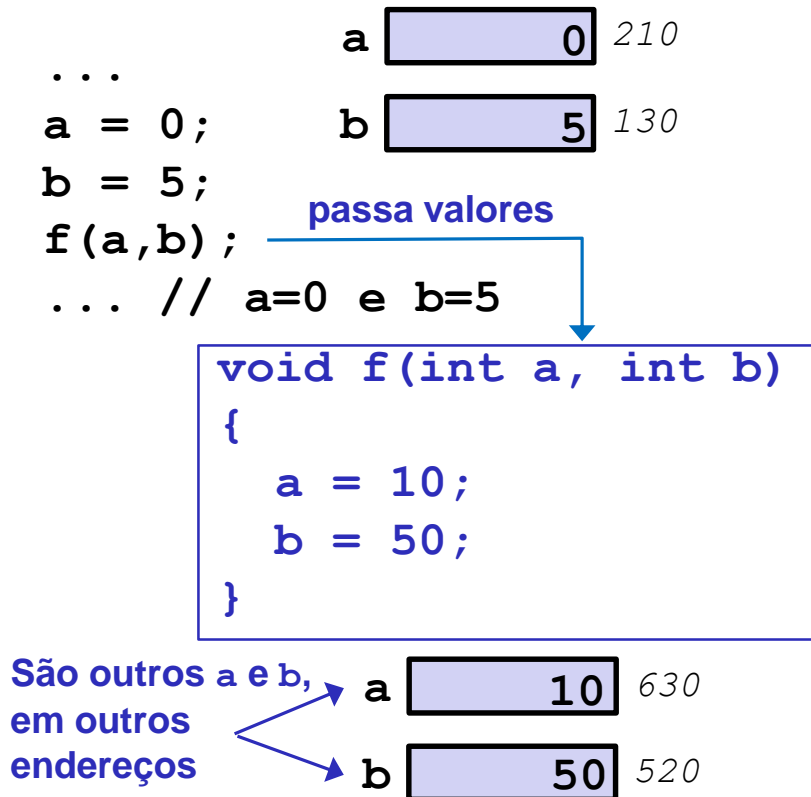
```
int main(void)
{
    int a, b, c, * p;
    a = 2;
    p = &c;
    *p = 3;
    b = a + (*p);
    printf(" %d ", b);
    return 0;
}
```

– **Atribuição **\*p = 3****

- **p aponta para c**
- **atribuição armazena 3 no espaço de memória reservado para c**

# Passagem de ponteiros para funções

- Chamar uma função *f* por valor (*call by value*) não modifica as variáveis.
- Passar valores de endereços de memória para uma função *f* permite a modificação dos valores das variáveis dentro de *f*.





# Passagem de ponteiros para funções – Exemplo 1

```
#include <stdio.h>
int power(int, int); // prototipo

int main(void)
{
    int n = 3;
    printf("%d\n", power(2,n));
    printf("%d\n", n);
    return 0;
}

int power(int base, int n)
{
    int p;
    for (p=1; n>0; n--)
        p*=base;
    return p;
}
```

Este exemplo ilustra o fato de que os argumentos passados para uma função não sofrem alterações (na realidade são passadas cópias destes argumentos). Neste exemplo, n não é destruído !

base	n	p
2	3	1
2	2	2
2	1	4
2	0	8

# Passagem de ponteiros para funções - Exemplo

Neste exemplo, vamos usar protótipo de função para poder colocar a função depois da `main`.

```
#include <stdio.h>
void troca(int * px, int * py); // prototipo

int main(void)
{
    int a = 5, b = 7;
    troca(&a, &b);      /* passamos os endereços das variáveis */
    printf("%d %d \n", a, b);
    return 0;
}

void troca(int * px, int * py )
{
    int temp;
    temp = *px;
    *px = *py;
    *py = temp;
}
```

O protótipo pode ter apenas as declarações:  
`void troca(int *, int *)`

# Ponteiros como Argumentos de Função - Exemplo

Escreva função `cone` que retorna void e calcula área total e volume de um cone reto

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

#define PI 3.14159265f

void cone(float r, float h, float * area, float * volume)
{
    float s = sqrt(r * r + h * h); // melhor: float s = (float)sqrt(...);
    *area = PI * r * (r + s); // base + lateral
    *volume = (PI * r * r * h) / 3.0f;
    return;
}

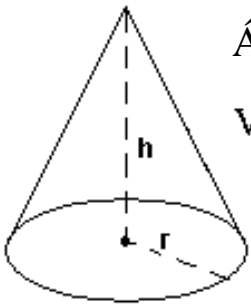
int main(void)
{
    float area, volume;
    float r = 2, h = 5;
    cone(r, h, &area, &volume);
    printf("Area=%f Volume=%f\n", area, volume);
    return 0;
}
```

*sem o f é double*

*base + lateral*

Área total =  $\pi r (r + \sqrt{r^2 + h^2})$

Volume =  $\frac{1}{3} \pi r^2 h$



A diagram of a right circular cone. The base is a circle with radius 'r' indicated by a dashed line from the center to the edge. The height 'h' is indicated by a dashed vertical line from the apex to the center of the base. The cone is shown in a 3D perspective.

```
C:\WINDOWS\system32\cmd.exe
Area=46.402359 Volume=20.943953
Press any key to continue . . .
```

# Usando módulos

**geometria.h**  
(header file)

Escreva a função cone anterior, usando módulos.

*protótipo*

```
#define PI 3.14159265f
void cone(float r, float h, float * area, float * volume);
```

**geometria.c**  
(source file)

*o seu .h local  
deve estar  
entre aspas  
duplas "..."*

```
#include <math.h>
#include "geometria.h"
void cone(float r, float h, float * area, float * volume)
{
    float s = sqrt( r * r + h* h );
    *area = PI * r * (r + s);
    *volume = (PI * r * r * h ) / 3.0f ;
    return;
}
```

**prog.c**  
(source file)

*note que neste  
módulo você  
não precisa de  
math.h*

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

int main(void)
{
    float area, volume;
    float r=2, h=5;
    cone(r,h,&area,&volume);
    printf("Area=%f Volume=%f\n",area,volume);
    return 0;
}
```

SolucaoComModulos

- References
- External Dependencies
- Header Files
  - geometria.h
- Resource Files
- Source Files
  - geometria.c
  - prog.c

Right-click **Header Files**  
Add > New item...

Right-click **Source Files**  
Add > New item...