

# LÓGICA DE PROGRAMAÇÃO

PROF<sup>a</sup>. M.Sc. JULIANA H Q BENACCHIO

- 1) Funções simples
- 2) Funções com passagem de parâmetro por valor
- 3) Funções com passagem de parâmetro por referência

- Em geral, podem ser passados argumentos para funções de duas maneiras:
  - Chamada por valor, onde o valor de um argumento é copiado no parâmetro formal da função;
  - Chamada por referência, onde o endereço de um argumento é copiado no parâmetro;

- Na passagem de parâmetro por referência, o endereço da variável é usado para acessar o argumento real utilizado na chamada.
- Isso significa que alterações feitas no parâmetro afetam a variável usada para chamar a função.
- Para isto, precisamos utilizar um novo conceito: **ponteiro**.

- Uma variável do tipo `int` armazena valores inteiros.
  - Uma variável do tipo `float` armazena números de ponto flutuante.
  - Uma variável do tipo `char` armazena caracteres.
- **Ponteiros armazenam endereços de memória.**

- Ponteiros são usados em situações em que é necessário conhecer o endereço onde está armazenada a variável e não o seu conteúdo.
- Um ponteiro é uma variável que contém um endereço de memória e não o conteúdo da posição.

- A memória de um computador pode ser vista como uma sequência de bytes cada um com seu próprio endereço.
- A tabela abaixo mostra um mapa de um trecho de memória que contém duas variáveis:

```
int num=10, res=120;
```

Endereços	Conteúdo	Variável
999	---	---
1000	10	num
1001	120	res

# Operações com Ponteiros

- **Declaração:**
- Antes de serem usados os ponteiros, como as variáveis, precisam ser declarados. A forma geral da declaração de um ponteiro é a seguinte:  
  
`tipo *nome;`
- Onde tipo é qualquer tipo válido em C e nome é o nome da variável ponteiro. Por exemplo:

```
int *px; //ponteiro para uma variavel inteira
```

```
char *ch; //ponteiro para uma variavel char
```



# Operações com Ponteiros

- **Operador de endereço &:**
- O operador & devolve o endereço de memória do seu operando. Por exemplo,

```
pres = &soma; //pres recebe o endereço da variável soma
```

- No exemplo seguinte considere a tabela de memória. Após a execução do trecho de programa abaixo a variável ponteiro p termina com o valor 1000.

```
p = &num;
```

# Operações com Ponteiros

- **Operador indireto \***
- O operador \* é o complemento de &. O operador \* devolve o valor da variável localizada no endereço que o segue.

Não confundir:

O operador \* para ponteiros não tem nada a ver com o operador de multiplicação.

# Operações com Ponteiros

- Por exemplo, o comando

```
num = *p;
```

//num recebe o conteúdo da variável localizada no endereço de p

- significa que a variável num recebe o valor apontado por p.

# Exemplo

```
int main()
{
    int num, valor;

    int *p;

    num = 55;

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

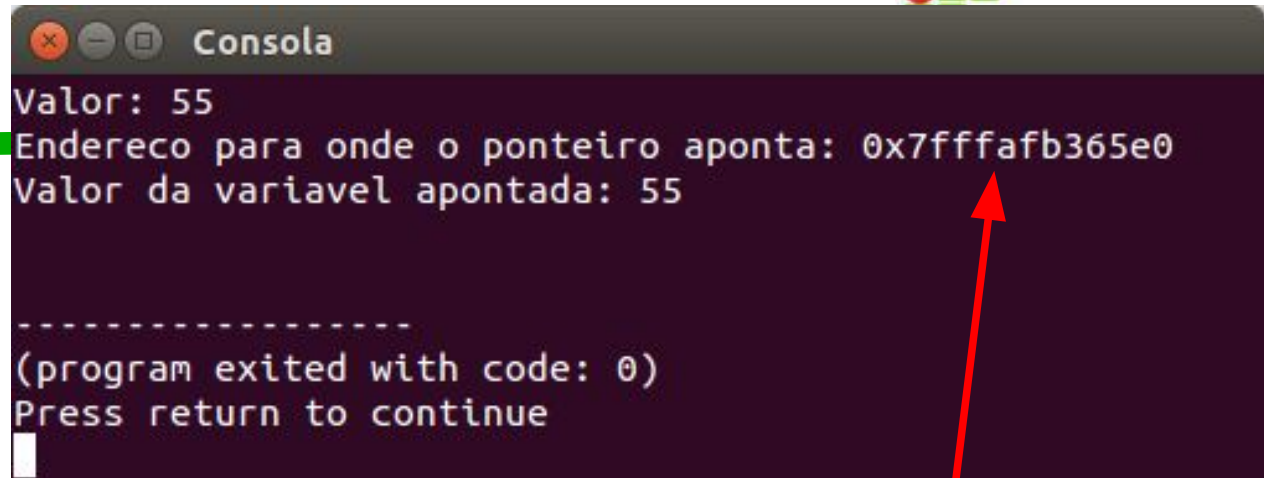
    valor = *p; // Valor é igualado a num de uma maneira indireta

    printf("Valor: %d\n", valor);

    printf("Endereco para onde o ponteiro aponta: %p\n", p);

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

    return 0;
}
```



```
Consola
Valor: 55
Endereco para onde o ponteiro aponta: 0x7ffffafb365e0
Valor da variavel apontada: 55

-----
(program exited with code: 0)
Press return to continue
```

ou algum número  
em hexadecimal que  
representa o endereço

# Exemplo

```
int main( )
{
    int num;

    int *p;

    num = 55;

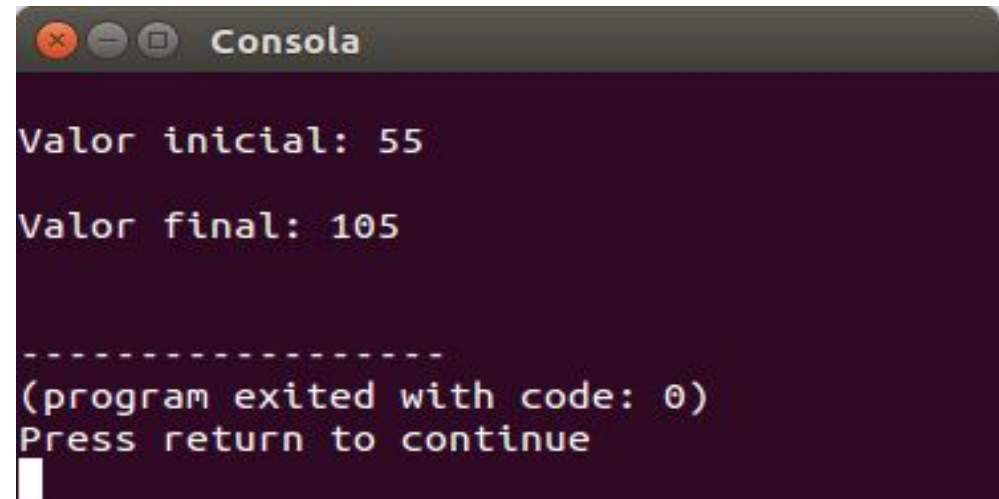
    p = &num; // Pega o endereço de num

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

    *p=105; // Muda o valor de num de uma maneira indireta

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

    return 0;
}
```



```
Consola

Valor inicial: 55

Valor final: 105

-----
(program exited with code: 0)
Press return to continue
```

# Funções com passagem de parâmetro por referência

- Suponha que precisamos de uma função que troque os valores de duas variáveis inteiras, digamos a e b.

```
void troca(int a, int b)
{
    int temp;
    temp = a;
    a = b;
    b = temp;
}
```

# Funções com passagem de parâmetro por referência

```
int main()
{
    int a = 2, b = 3;

    printf("Antes de chamar a função : \na=%d\nb=%d\n", a, b);

    troca(a, b);

    printf("Depois de chamar a função: \na=%d\nb=%d\n", a, b);

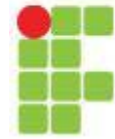
    return 0;
}
```

# Funções com passagem de parâmetro por referência

- A função não produz o efeito desejado, pois recebe apenas os valores das variáveis e não as variáveis propriamente ditas.
- A função recebe "cópias" das variáveis e troca os valores dessas cópias, enquanto as variáveis "originais" permanecem inalteradas.
- Para obter o efeito desejado, é preciso passar à função os **endereços** das variáveis



# Funções com passagem de parâmetro por referência



```
void troca(int *a, int *b)
{
    int temp;

    temp = *a;

    *a = *b;

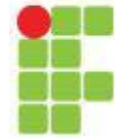
    *b = temp;
}
```

# Funções com passagem de parâmetro por referência

- Função que recebe duas variáveis inteiras e "zera" o valor das variáveis.

```
void zera(int *a, int *b)
{
    *a = 0;
    *b = 0;
}
```

# Funções com passagem de parâmetro por referência



INSTITUTO FEDERAL  
PARANÁ

```
int main()
{
    int a = 2, b = 3;

    printf("Antes de chamar a funcao : \na=%d\nb=%d\n", a, b);

    troca(&a, &b);

    printf("Depois de chamar a funcao: \na=%d\nb=%d\n", a, b);

    return 0;
}
```

# Protótipos de Funções

- A forma geral de uma definição de protótipo de função é:

```
tipo nome_da_função (lista_de_argumentos);
```

- Protótipos são **declarações de funções**. Isto é, você declara uma função que irá usar. O compilador toma então conhecimento do formato daquela função antes de compilá-la.

# Protótipos de Funções

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

```
float quadrado (float a);
```

```
float cubo (float a);
```

```
int main () {
```

```
    float num;
```

```
    printf("Entre com um numero: ");
```

```
    scanf("%f", &num);
```

```
    printf ("Numero elevado ao quadrado= %.2f\n", quadrado(num));
```

```
    printf ("Numero elevado ao cubo = %.2f\n", cubo(num));
```

```
    return 0;
```

```
}
```

# Protótipos de Funções

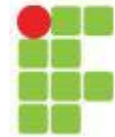
```
float quadrado (float a)
```

```
{  
    return (a*a);  
}
```

```
float cubo (float a)
```

```
{  
    return (a*a*a);  
}
```

# Chamando funções com Matrizes



INSTITUTO FEDERAL  
PARANÁ

- Para uma matriz (ou vetor) ser usada como um argumento para uma função, apenas o endereço da matriz deve ser passado, e não uma cópia da matriz inteira.
- Quando você chama uma função com um nome de matriz, um ponteiro para o primeiro elemento na matriz é passado para a função.



O nome da matriz sem qualquer índice é um ponteiro para o primeiro elemento na matriz

# Chamando funções com Matrizes

- Por exemplo, para o vetor:

```
int vetor [50];
```

- A declaração da func() pode ser feita de três maneiras:

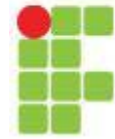
```
void func (int vet [50] );
```

```
void func (int vet [] );
```

```
void func (int *vet);
```



# Chamando funções com Matrizes



INSTITUTO FEDERAL  
PARANÁ

- Nos três casos, teremos dentro de `func()` um `int*` chamado `vet`.
- Ao passarmos um vetor para uma função, na realidade estamos passando um ponteiro.
- Neste ponteiro é armazenado o endereço do primeiro elemento do vetor. Isto significa que não é feita uma cópia, elemento a elemento do vetor. Isto faz com que possamos alterar o valor dos elementos do vetor dentro da função.