

Aula 12:

Funções

Disciplina: Fundamentos de Programação

Prof. Luiz Olmes

olmes@unifei.edu.br



UNIFEI
UNIVERSIDADE FEDERAL DE ITAJUBÁ

Nas aulas anteriores...

▶ **O QUE JÁ ESTUDAMOS?**

- ▶ Algoritmos.
- ▶ Linguagem C.
- ▶ Variáveis, operadores e tipos.
- ▶ Estruturas de controle condicionais.
- ▶ Estruturas de controle iterativas.
- ▶ Vetores.
- ▶ Matrizes.
- ▶ Strings.
- ▶ Estruturas.
- ▶ Ponteiros e Alocação Dinâmica.

▶ **OBJETIVOS:**

- ▶ Definição.
- ▶ Protótipo.
- ▶ Fluxo de controle de execução.
- ▶ Passagem de parâmetros:
 - ▶ Por valor
 - ▶ Por referência

Contexto

- ▶ A modularização é um recurso muito importante apresentado nas linguagens de programação:
 - ▶ Divisão do programa em módulos específicos.
- ▶ A linguagem C possibilita a modularização de código por meio de **funções** (também chamadas de **métodos** ou **sub-rotinas**).
- ▶ Programas em C são escritos combinando funções pertencentes à linguagem com aquelas definidas pelo programador.
- ▶ O programador é livre para criar quantas funções forem necessárias, dependendo do problema que está sendo resolvido.

Funções: Definição

- ▶ Uma **função** é um agrupamento de comandos que realizam uma **tarefa específica**.
- ▶ Funções são utilizadas para:
 - ▶ **Modularizar o código**: o programa é construído a partir de pequenos blocos de código (funções) com finalidades específicas.
 - ▶ **Promover a reutilização de código**: o código de uma função existe em um único ponto do programa, embora sua chamada ocorra em vários.
- ▶ Cada função realiza uma única tarefa. Ao finalizar, retorna-se para o ponto onde a função foi invocada pelo programa principal ou por outra função.

Funções

- ▶ Funções não podem ser aninhadas:
 - ▶ Uma função não pode ser declarada dentro de outra função, apenas invocada por outra função.
- ▶ Todo código C possui, no mínimo, uma função chamada `main()`, que é por onde a execução do programa se inicia.
- ▶ C traz diversas funções implementadas em suas bibliotecas:
 - ▶ `printf`, `scanf`: pertencentes ao arquivo `stdio.h`
 - ▶ `system`, `malloc`, `calloc`, `free`, `rand`: pertencentes ao arquivo `stdlib.h`
 - ▶ `sqrt`, `pow`, `sin`, `cos`: pertencentes ao arquivo `math.h`
 - ▶ `strlen`, `strcpy`, `strcmp`: pertencentes ao arquivo `string.h`

Funções

► Sintaxe:

```
tipo nomeDaFunção(listaDeParametros)
{
    ...
    return expressao;
}
```

Funções

► Sintaxe:

```
tipo nomeDaFunção(listaDeParametros)
{
    ...
    return expressao;
}
```

- **tipo**: tipo ou valor devolvido pela função (int, char, float, ...).
 - Em C, a função `main` é declarado `int main()` e devolve um inteiro.
- **nomeDaFunção**: identificador ou nome dado à função.
- **listaDeParametros**: são as variáveis passadas à função.
- **expressao**: indica o valor devolvido ao finalizar.

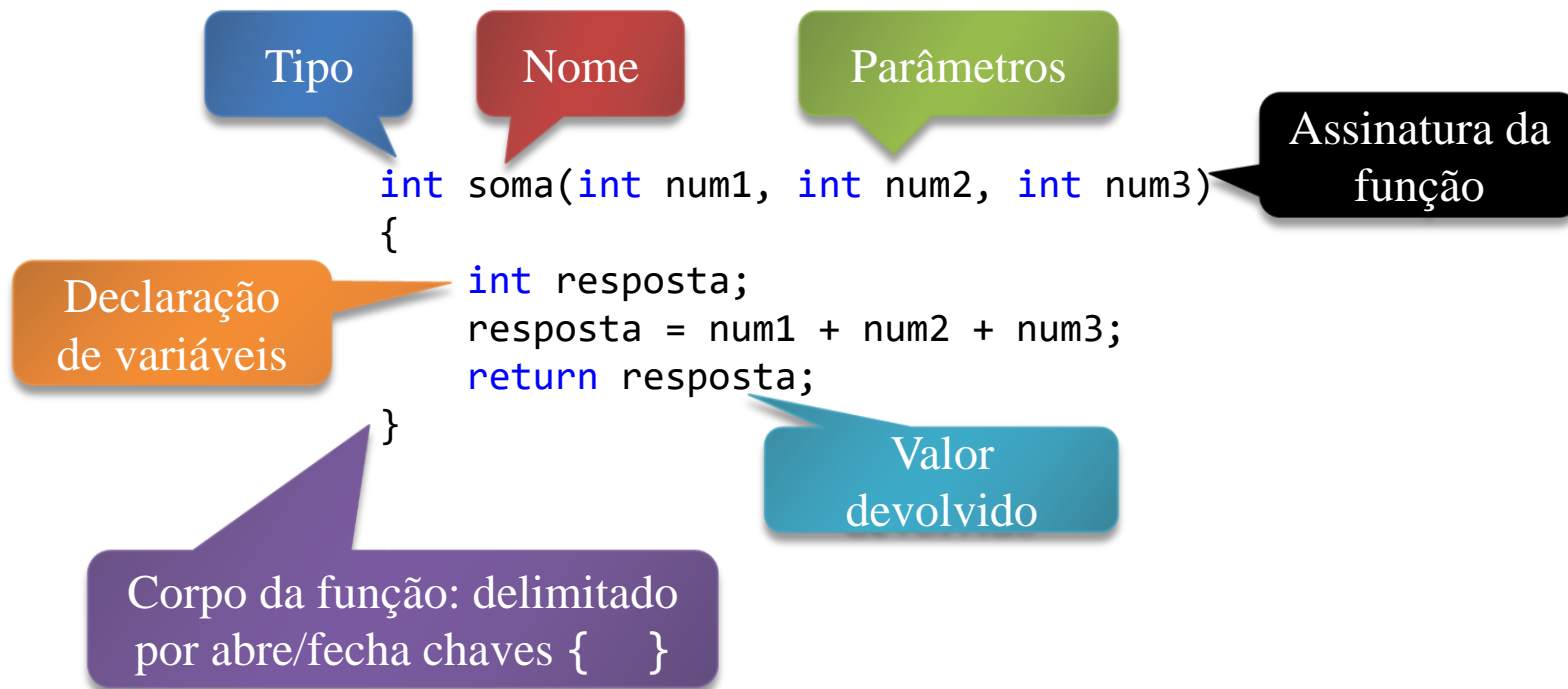
Funções

- ▶ **Exemplo:** função que soma 3 valores inteiros.

```
int soma(int num1, int num2, int num3)
{
    int resposta;
    resposta = num1 + num2 + num3;
    return resposta;
}
```


Funções

- **Exemplo:** função que soma 3 valores inteiros.



Protótipos de funções

- ▶ Em C, é necessário que uma função seja **declarada** ou **definida** antes do seu uso.
- ▶ A declaração de uma função sem a sua implementação, antes do `main()`, é chamada de **protótipo** da função.
- ▶ O compilador utiliza protótipos de funções para **validar** se a função foi invocada corretamente, com os números e tipos de parâmetros corretos e tipo de retorno adequado.
 - ▶ Se houver alguma inconsistência, uma mensagem de erro é retornada.

Exemplo: função que eleva ao cubo

```
1. #include <stdio.h>
2.
3. int cubo(int n);
4.
5. int main()
6. {
7.     int x = 2, y;
8.     y = cubo(x);
9.
10.    printf("Y = %d \n", y);
11.    return 0;
12.}
13.
14.int cubo(int n)
15.{
16.    return (n * n * n);
17.}
```

Protótipo da função

Implementação da
função depois do
main()

Fluxo de execução do programa

- ▶ O código de um programa é executado até **encontrar** uma **função**.
- ▶ O programa é então interrompido temporariamente, e o **fluxo** de **execução** do programa **passa para** a **função** chamada.
- ▶ Se houver parâmetros na função, os valores da chamada da função são copiados para os parâmetros no código da função.
- ▶ Os comandos da **função** são **executados**.
- ▶ Quando a função termina, o programa volta ao **ponto** em que foi **interrompido** para continuar sua execução normal.
- ▶ Se houver um comando **return**, o valor dele será copiado para a **variável** que foi escolhida para receber o retorno da função.

Fluxo de execução do programa

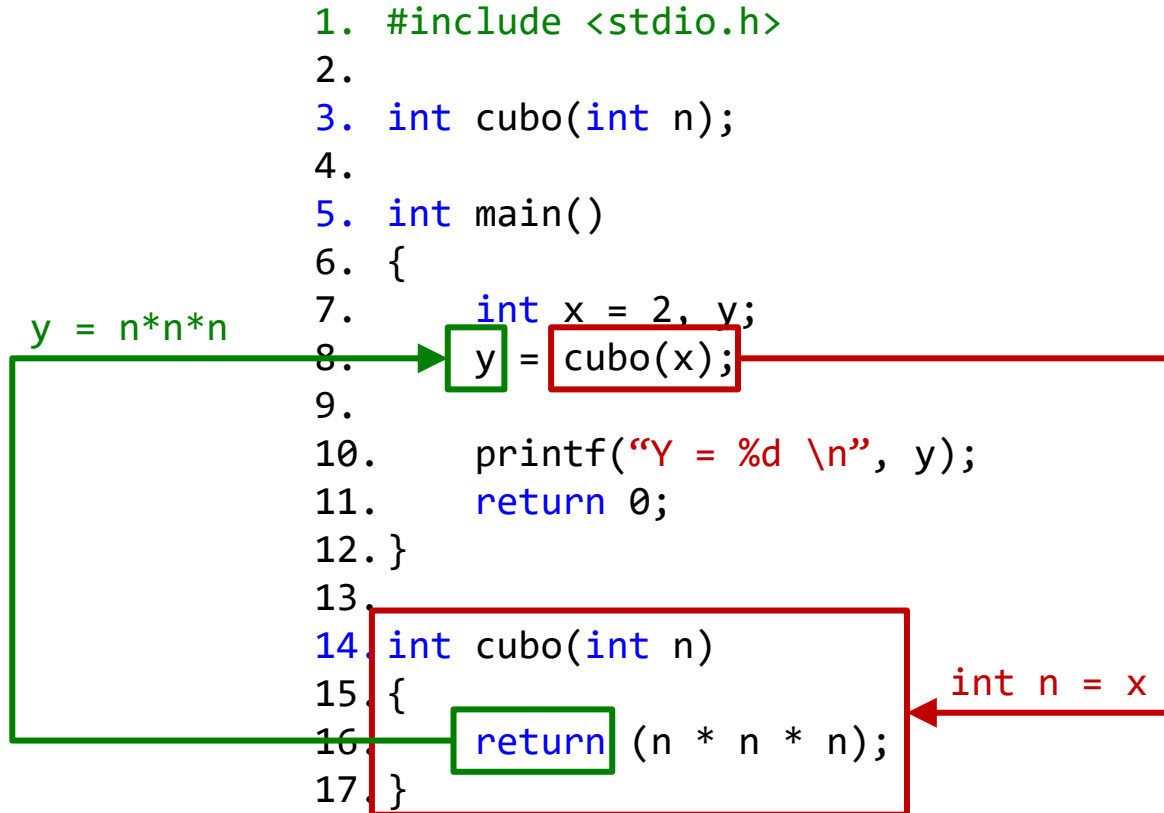
```
1. #include <stdio.h>
2.
3. int cubo(int n);
4.
5. int main()
6. {
7.     int x = 2, y;
8.     y = cubo(x);
9.
10.    printf("Y = %d \n", y);
11.    return 0;
12.}
13.
14.int cubo(int n)
15.{
16.    return (n * n * n);
17.}
```

Fluxo de execução do programa

```
1. #include <stdio.h>
2.
3. int cubo(int n);
4.
5. int main()
6. {
7.     int x = 2, y;
8.     y = cubo(x);
9.
10.    printf("Y = %d \n", y);
11.    return 0;
12.}
13.
14. int cubo(int n)
15. {
16.     return (n * n * n);
17.}
```

The diagram illustrates the execution flow of the program. A red box highlights the function call `cubo(x)` on line 8. A red arrow originates from this box and points to the parameter `n` in the function definition `int cubo(int n)` on line 14. Another red box highlights the entire function definition on lines 14-17. A red arrow points from the box on line 8 to the box on line 14, indicating the transfer of control to the function. The text `int n = x` is written in red next to the arrow, indicating the argument passing.

Fluxo de execução do programa



Passagem de parâmetros

- ▶ A linguagem C permite que funções recebam ou não parâmetros.
- ▶ **Funções sem parâmetros**: declaradas sem nenhuma variável dentro dos parênteses, ou com a palavra **void** entre eles.
 - ▶ tipo funcao01()...
 - ▶ tipo funcao02(void)...

Passagem de parâmetros

- ▶ A linguagem C permite que funções recebam ou não parâmetros.
- ▶ **Funções sem parâmetros**: declaradas sem nenhuma variável dentro dos parênteses, ou com a palavra **void** entre eles.
 - ▶ `tipo funcao01()...`
 - ▶ `tipo funcao02(void)...`
- ▶ **Funções que recebem parâmetros**: a passagem de parâmetros para funções pode ser realizada de dois modos:
 - ▶ Passagem por **valor**.
 - ▶ Passagem por **referência** (*próxima aula*).

Passagem de parâmetros por valor

- ▶ Também conhecida como **passagem por cópia**.
- ▶ Quando o código é compilado e a função invocada:
- ▶ A função recebe uma **cópia** dos valores dos parâmetros.
- ▶ Se o valor de um parâmetro é alterado no corpo da função, a **mudança afeta apenas a função, e não tem efeito fora dela**.

Funções

- ▶ **Exemplo 1:** escrever uma função que receba três números inteiros e retorne o maior valor.

Funções

- ▶ **Exemplo 1:** escrever uma função que receba três números inteiros e retorne o maior valor.

O que é necessário definir?

Funções

- ▶ **Exemplo 1:** escrever uma função que receba três números inteiros e retorne o maior valor.

O que é necessário definir?

Tipo

Nome

Parâmetros

```
tipo nomeDaFunção(listaDeParametros)
{
    ...
    return expressao;
}
```

Valor
devolvido

Funções

- ▶ **Exemplo 1:** escrever uma função que receba três números inteiros e retorne o maior valor.

O que é necessário definir?

Tipo

Nome

Parâmetros

```
tipo maximo(listaDeParametros)
{
    ...
    return expressao;
}
```

Valor
devolvido

Funções

- **Exemplo 1:** escrever uma função que **receba três números inteiros** e retorne o maior valor.

O que é necessário definir?

Tipo

Nome

Parâmetros

```
tipo maximo(int a, int b, int c)
{
    ...
    return expressao;
}
```

Valor
devolvido

Funções

- ▶ **Exemplo 1:** escrever uma função que receba três números inteiros e **retorne o maior valor**.

O que é necessário definir?

Tipo

Nome

Parâmetros

```
int maximo(int a, int b, int c)
{
    ...
    return expressao;
}
```

Valor
devolvido

Funções

- ▶ **Exemplo 1:** escrever uma função que receba três números inteiros e retorne o maior valor.

O que é necessário definir?

Tipo

Nome

Parâmetros

```
int maximo(int a, int b, int c)
{
    ...
    return expressao;
}
```

Valor devolvido: variável ou expressão do tipo `int`, de acordo com a implementação

Funções

- **Exemplo 1:** escrever uma função que receba três números inteiros e retorne o maior valor.

```
1. #include <stdio.h>
2.
3. int maximo(int a, int b, int c); // protótipo
4.
5. int main()
6. {
7.     int x, y, z, m;
8.
9.     printf("Digite 3 numeros:\n");
10.    scanf("%d %d %d", &x, &y, &z);
11.
12.    m = maximo(x, y, z);
13.    printf("Maior valor = %d \n", m);
14.
15.    return 0;
16. }
```

// CONTINUA APÓS O MAIN...

Funções

- **Exemplo 1:** escrever uma função que receba três números inteiros e retorne o maior valor.

```
17.  
18.  
19.  
20.  
21.  
22.  
23.  
24. int maximo(int a, int b, int c)  
25. {  
26.     int max = a; // assume que 'a' eh o maior valor  
27.  
28.     if(b > max) max = b; // se 'b' for maior, troca  
29.     if(c > max) max = c; // se 'c' for maior, troca  
30.  
31.     return max; // devolve o maior valor  
32. }
```

Funções

- **Exemplo 1:** escrever uma função que receba três números inteiros e retorne o maior valor.

```
17. /**
18.  * Funcao que retorna o valor maximo de tres inteiros.
19.  * @param a 0 primeiro valor inteiro.
20.  * @param b 0 segundo valor inteiro.
21.  * @param c 0 terceiro valor inteiro.
22.  * @return 0 valor maximo entre a, b e c.
23. */
24. int maximo(int a, int b, int c)
25. {
26.     int max = a; // assume que 'a' eh o maior valor
27.
28.     if(b > max) max = b; // se 'b' for maior, troca
29.     if(c > max) max = c; // se 'c' for maior, troca
30.
31.     return max; // devolve o maior valor
32. }
```

Documentação
da função

Funções

- ▶ **Exemplo 2:** escrever uma função que receba um inteiro entre 1 e 7 e imprima, por extenso, o dia da semana correspondente.

Funções

- ▶ **Exemplo 2:** escrever uma função que receba um inteiro entre 1 e 7 e imprima, por extenso, o dia da semana correspondente.
- ▶ *Definições:*
 - ▶ **Tipo:** `void`, pois a função não devolve nenhum valor, apenas imprime.
 - ▶ **Nome:** `imprimeDia`
 - ▶ **Parâmetros:** um valor inteiro (`int`).

Funções

- **Exemplo 2:** escrever uma função que receba um inteiro entre 1 e 7 e imprima, por extenso, o dia da semana correspondente.

```
1. #include <stdio.h>
2.
3. void imprimeDia(int dia); // protótipo
4.
5. int main()
6. {
7.     int x;
8.
9.     printf("Digite o dia (1 a 7): ");
10.    scanf("%d", &x);
11.
12.    printf("Por extenso: ");
13.    imprimeDia(x);
14.
15.    return 0;
16. }
```

// CONTINUA APÓS O MAIN...

Funções

- ▶ **Exemplo 2:** escrever uma função que receba um inteiro entre 1 e 7 e imprima, por extenso, o dia da semana correspondente.

```
17.  
18. /**  
19.  * Funcao que imprime por extenso o nome do dia da semana.  
20.  * @param dia 0 inteiro que representa o dia.  
21.  */  
22. void imprimeDia(int dia)  
23. {  
24.     if(dia == 1) printf("Domingo\n");  
25.     if(dia == 2) printf("Segunda-feira\n");  
26.     if(dia == 3) printf("Terca-feira\n");  
27.     // continua...  
28.     if(dia == 7) printf("Sabado\n");  
29. }
```


Passagem de parâmetros por referência

- ▶ Por convenção, C é uma linguagem de passagem de parâmetros por valor.
- ▶ **Vantagem:** qualquer expressão pode ser usada como argumento de funções, respeitando apenas a tipagem da linguagem.
- ▶ **Desvantagens:**
 - ▶ Alto custo computacional em passar objetos de maior tamanho para funções.
 - ▶ Parâmetros não podem ser modificados, visto que são apenas cópias.

Motivação

- ▶ **Exemplo:** criar um programa que leia os valores de duas variáveis inteiras A e B e **uma função** que realize a permutação dos valores lidos, isto é, troque os seus valores.

Motivação

- ▶ **Exemplo:** criar um programa que leia os valores de duas variáveis inteiras A e B e **uma função** que realize a permutação dos valores lidos, isto é, troque os seus valores.

```
1. #include <stdio.h>
2. void troca(int a, int b);
3. int main()
4. {
5.     int a, b;
6.     printf("Digite A e B:\n");
7.     scanf("%d %d", &a, &b);
8.     printf("A e B antes: %d %d \n", a, b);
9.     troca(a, b);
10.    printf("A e B depois: %d %d \n", a, b);
11.    return 0;
12. }
13.
```

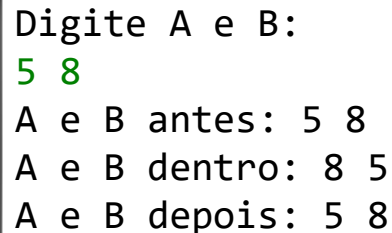
```
14. void troca(int a, int b)
15. {
16.     int temp = a;
17.     a = b;
18.     b = temp;
19.     printf("A e B dentro:
           %d %d \n", a, b);
20. }
```

Motivação

- ▶ **Exemplo:** criar um programa que leia os valores de duas variáveis inteiras A e B e **uma função** que realize a permutação dos valores lidos, isto é, troque os seus valores.

```
1. #include <stdio.h>
2. void troca(int a, int b);
3. int main()
4. {
5.     int a, b;
6.     printf("Digite A e B:\n");
7.     scanf("%d %d", &a, &b);
8.     printf("A e B antes: %d %d \n", a, b);
9.     troca(a, b);
10.    printf("A e B depois: %d %d \n", a, b);
11.    return 0;
12. }
13.
```

```
14. void troca(int a, int b)
15. {
16.     int temp = a;
17.     a = b;
18.     b = temp;
19.     printf("A e B dentro:
           %d %d \n", a, b);
20. }
```



```
Digite A e B:
5 8
A e B antes: 5 8
A e B dentro: 8 5
A e B depois: 5 8
```

Motivação

- ▶ **Exemplo:** criar um programa que leia os valores de duas variáveis inteiras A e B e **uma função** que realize a permutação dos valores lidos, isto é, troque os seus valores.

```
1. #include <stdio.h>
2. void troca(int a, int b);
3. int main()
4. {
5.     int a, b;
6.     printf("Digite A e B:\n");
7.     scanf("%d %d", &a, &b);
8.     printf("A e B antes: %d %d \n", a, b);
9.     troca(a, b);
10.    printf("A e B depois: %d %d \n", a, b);
11.    return 0;
12. }
13.
```

```
14. void troca(int a, int b)
15. {
16.     int temp = a;
17.     a = b;
18.     b = temp;
19.     printf("A e B dentro:
           %d %d \n", a, b);
20. }
```

Não modifica
variáveis externas!

Passagem de parâmetros por referência

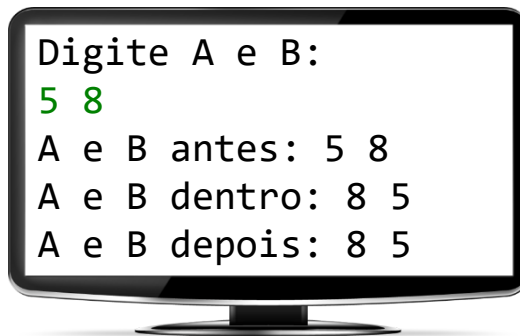
- ▶ Entretanto, algumas situações podem exigir que **variáveis externas** à função sejam alteradas e a alteração preservada.
- ▶ Por exemplo: `scanf()`
 - ▶ Lê dados e os armazena nas variáveis recebidas como parâmetros.
- ▶ Para que alterações em funções se reflitam fora de seu escopo, basta que os argumentos recebidos sejam endereços de memória.
- ▶ As variáveis são, então, passadas por **referência**.

Passagem de parâmetros por referência

```
1. #include <stdio.h>
2. void troca(int *a, int *b);
3. int main()
4. {
5.     int a = 5, b = 8;
6.     printf("A e B antes: %d %d \n", a, b);
7.     troca(&a, &b);
8.     printf("A e B depois: %d %d \n", a, b);
9.     return 0;
10.}
11.
12.void troca(int *a, int *b)
13.{
14.    int temp = *a;
15.    *a = *b;
16.    *b = temp;
17.    printf("A e B dentro: %d %d \n", *a, *b);
18.}
```

Passagem de parâmetros por referência

```
1. #include <stdio.h>
2. void troca(int *a, int *b);
3. int main()
4. {
5.     int a = 5, b = 8;
6.     printf("A e B antes: %d %d \n", a, b);
7.     troca(&a, &b);
8.     printf("A e B depois: %d %d \n", a, b);
9.     return 0;
10.}
11.
12. void troca(int *a, int *b)
13. {
14.     int temp = *a;
15.     *a = *b;
16.     *b = temp;
17.     printf("A e B dentro: %d %d \n", *a, *b);
18. }
```



Passagem de parâmetros por referência

```
1. #include <stdio.h>
2. void troca(int *a, int *b);
3. int main()
4. {
5.     int a = 5, b = 8;
6.     printf("A e B antes: %d %d \n", a, b);
7.     troca(&a, &b);
8.     printf("A e B depois: %d %d \n", a, b);
9.     return 0;
10.}
11.
12. void troca(int *a, int *b)
13. {
14.     int temp = *a;
15.     *a = *b;
16.     *b = temp;
17.     printf("A e B dentro: %d %d \n", *a, *b);
18. }
```

Função recebe
ponteiros.

Passagem de parâmetros por referência

```
1. #include <stdio.h>
2. void troca(int *a, int *b);
3. int main()
4. {
5.     int a = 5, b = 8;
6.     printf("A e B antes: %d %d \n", a, b);
7.     troca(&a, &b);
8.     printf("A e B depois: %d %d \n", a, b);
9.     return 0;
10.}
11.
12. void troca(int *a, int *b)
13. {
14.     int temp = *a;
15.     *a = *b;
16.     *b = temp;
17.     printf("A e B dentro: %d %d \n", *a, *b);
18. }
```

Endereços das variáveis
são repassados.

Função recebe
ponteiros.

Passagem de parâmetros por referência

```
1. #include <stdio.h>
2. void troca(int *a, int *b);
3. int main()
4. {
5.     int a = 5, b = 8;
6.     printf("A e B antes: %d %d \n", a, b);
7.     troca(&a, &b);
8.     printf("A e B depois: %d %d \n", a, b);
9.     return 0;
10.}
11.
12. void troca(int *a, int *b)
13. {
14.     int temp = *a;
15.     *a = *b;
16.     *b = temp;
17.     printf("A e B dentro: %d %d \n", *a, *b);
18. }
```

Endereços das variáveis
são repassados.

Função recebe
ponteiros.

Alterações persistem
fora da função.

Passagem de parâmetros por referência

- ▶ A passagem de parâmetros por referência também pode ser usada para casos onde a função “devolve” **mais de um valor**.
- ▶ Nesse caso, as variáveis são recebidas por referência e têm o seus valores alterados no corpo do função.
- ▶ **Cuidado com o abuso deste recurso!** Lembre-se de uma das premissas de uma função, que é **realizar uma tarefa específica**.
- ▶ **Exemplo**: criar uma função que, dada a medida do lado de um quadrado, calcule o perímetro e a área do quadrado.

Passagem de parâmetros por referência

```
1. #include <stdio.h>
2. void quadrado(float lado, float *perimetro, float *area);
3.
4. int main()
5. {
6.     float lado, perimetro, area;
7.     printf("Digite a medida do lado: ");
8.     scanf("%f", &lado);
9.     quadrado(lado, &perimetro, &area);
10.    printf("Perim = %f Area = %f\n", perimetro, area);
11.    return 0;
12.}
13.
14.void quadrado(float lado, float *perimetro, float *area);
15.{
16.    *perimetro = 4 * lado;
17.    *area = lado * lado;
18.}
```

Retorno x passagem por referência

Usando retorno

```
1. #include <stdio.h>
2. int funcao();
3.
4. int main()
5. {
6.     int a = 1;
7.     printf("Antes:  a = %d\n", a);
8.     a = funcao();
9.     printf("Depois: a = %d\n", a);
10.    return 0;
11. }
12.
13. int funcao()
14. {
15.     return 5;
16. }
```

Passagem por referência

```
1. #include <stdio.h>
2. void funcao(int *x);
3.
4. int main()
5. {
6.     int a = 1;
7.     printf("Antes:  a = %d\n", a);
8.     funcao(&a);
9.     printf("Depois: a = %d\n", a);
10.    return 0;
11. }
12.
13. void funcao(int *x)
14. {
15.     *x = 5;
16. }
```

Prática

- ▶ **Exercício 1:** escreva uma função para calcular o MMC (mínimo múltiplo comum) entre dois valores inteiros.
- ▶ **Exercício 2:** escreva uma função para calcular o MDC (máximo divisor comum) entre dois valores inteiros.
- ▶ **Exercício 3:** escreva uma função que dado um número real passado como parâmetro, retorne a parte inteira e a parte fracionária deste número. Protótipo: `void frac(float num, int *inteira, float *fracionaria);`
- ▶ **Exercício 4:** escreva um programa, dividido em funções, que: leia um valor N (main), aloque um vetor de N posições (função), leia seus elementos (apenas inteiros – função), ordene seus elementos (função), imprima o vetor (função).

Dúvidas?



Aula 12:

Funções

Disciplina: Fundamentos de Programação

Prof. Luiz Olmes

olmes@unifei.edu.br



UNIFEI
UNIVERSIDADE FEDERAL DE ITAJUBÁ