



Professor  
Maromo

# LINGUAGEM DE PROGRAMAÇÃO

Material 006



GitHub  
maromo71

# C



# Agenda



## Funções

- **Conceito**
- **Importância**
- **Passagem de parâmetros por valor e referência**

**Material: LP\_006**



# Funções



A large, bold, black stylized representation of the mathematical notation for a function,  $f(x)$ , is positioned in the upper right area of the slide. The background of the slide features a faint, semi-transparent image of a computer screen displaying code snippets, including CSS class selectors like `.widget-area` and `float: right;`, and a taskbar at the bottom with various application icons.

Em linguagem C, uma **função** é um **grupo de instruções que realiza uma tarefa específica**. Ela é autocontida, ou seja, pode ser vista como uma **unidade independente que recebe alguns dados como entrada (através de argumentos) e retorna um resultado (um valor ou uma série de valores)**, ou pode **não retornar nada (procedimento)**. Funções são fundamentais para a modularização de código, permitindo a reutilização e facilitando a manutenção.



# Sintaxe

```
tipo_de_retorno nome_da_funcao(parametro1, parametro2, ..., parametroN) {  
    // Corpo da função  
    ...  
    return valor; // Se tipo_de_retorno não for void  
}
```

- **Tipo\_de\_retorno:** Este é o tipo de dado que a função vai retornar.
- **nome\_da\_funcao:** É o identificador da função. Deve ser único dentro do escopo do programa.
- **parametro1, parametro2, ..., parametroN:** São os parâmetros da função. Cada parâmetro é definido por um tipo e um nome (por exemplo, int x). A função pode ter nenhum (indicado por void), um ou vários parâmetros.
- **Corpo da função:** Delimitado por chaves **{ }**, contém as instruções que serão executadas quando a função for chamada.
- **return:** Esta é a instrução usada para retornar um valor da função.



# Considere o programa

Nele se recorre a três funções distintas, para escrever na tela a seguinte saída:

```
***
*****
*****
*****
***
```



As funções **linha3x()**, **linha5x()** e **linha7x()** são responsáveis individualmente pela escrita dos asteriscos na tela.

A função **main()** invoca todas as outras funções declaradas anteriormente.

```
1  #include <stdio.h>
2  void linha3x(){
3      for (int i = 0; i < 3; ++i) {
4          putchar('*');
5      }
6      putchar('\n');
7  }
8  void linha5x(){
9      for (int i = 0; i < 5; ++i) {
10         putchar('*');
11     }
12     putchar('\n');
13 }
14 void linha7x(){
15     for (int i = 0; i < 7; ++i) {
16         putchar('*');
17     }
18     putchar('\n');
19 };
20 int main() {
21     linha3x();
22     linha5x();
23     linha7x();
24     linha5x();
25     linha3x();
26     return 0;
27 }
```

## Nota



Observe que criamos três funções com tarefas similares:

- linha3x()
- linha5x()
- linha7x()

Poderíamos criar apenas uma função que recebe-se como parâmetro 1 número inteiro, referente ao número de asteriscos que devem ser escritos.



# Programa Melhorado


```
***
*****
*****
*****
***
```



```
1  #include <stdio.h>
2  void linha(int num){
3      for (int i = 0; i < num; ++i) {
4          putchar('*');
5      }
6      putchar('\n');
7  }
8  int main() {
9      linha(num: 3);
10     linha(num: 5);
11     linha(num: 7);
12     linha(num: 5);
13     linha(num: 3);
14     return 0;
15 }
```

Parâmetro: **num** do tipo inteiro. Significa que a função recebe como argumento um número inteiro passado à ela.

# Parâmetros



A comunicação com uma função se faz através de **argumentos** que são enviados e dos **parâmetros presentes na função** que os recebe.

O número de parâmetros pode ser **0,1,2...,N** depende das necessidades de codificação.

Cada função necessita saber qual o tipo de cada um dos parâmetros.



# Exemplo

## Cálculo do IMC

Vamos criar um programa simples em C que calcula o Índice de Massa Corporal (IMC) usando funções. O IMC é calculado pela fórmula:

$$\text{IMC} = \frac{\text{peso (em kg)}}{\text{altura (em m)}^2}$$



# Exemplo: IMC

A instrução **return** permite terminar a execução de uma função e voltar ao programa ou função que a invocou.

No exemplo a instrução além de terminar a função, **devolve um valor ao programa principal como resultado de sua execução.**

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

```
// Função para calcular o IMC
```

```
float calcularIMC(float peso, float altura) {  
    return peso / (altura * altura);  
}
```

```
int main() {
```

```
    float peso, altura, imc;
```

```
    // Entrada de dados
```

```
    printf("Informe o peso (em kg): ");
```

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

```
    printf("Informe a altura (em metros): ");
```

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

```
    // Cálculo do IMC usando a função
```

```
    imc = calcularIMC(peso, altura);
```

```
    // Exibição do resultado
```

```
    printf("Seu IMC é: %.2f\n", imc);
```

```
    return 0;
```

```
}
```

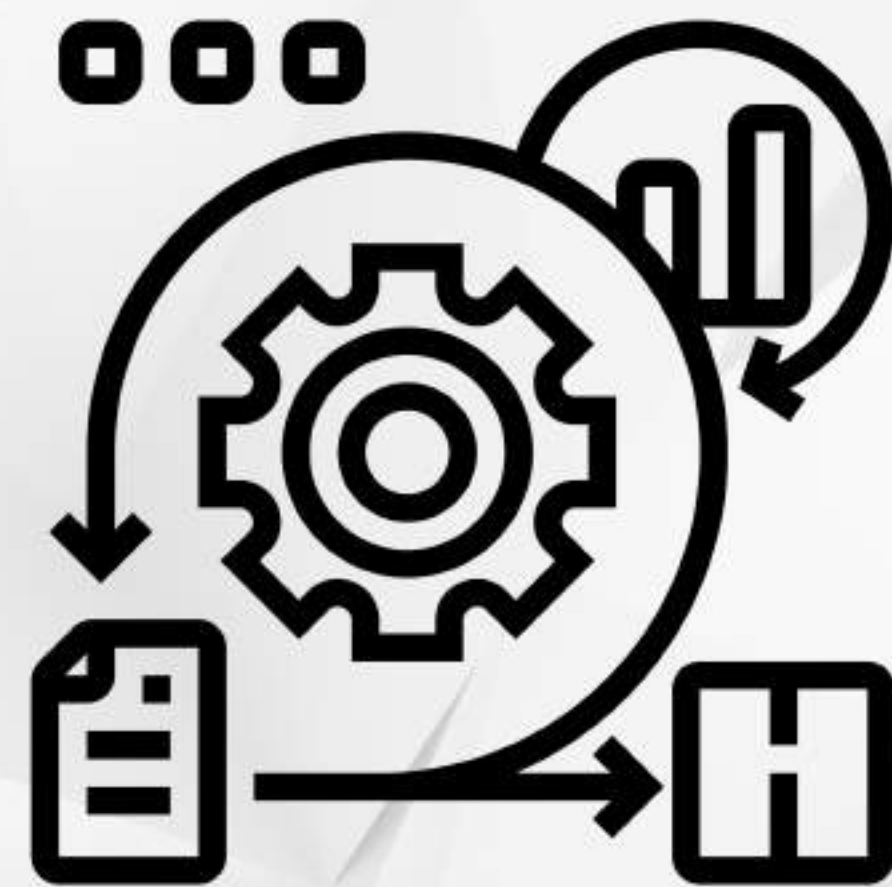


# Características de uma função



- Cada função deve **ter um nome único**.
- Uma função pode ser **invocada por outras funções**.
- Uma função deve realizar **UMA ÚNICA TAREFA** bem definida.
- Uma função deve comportar-se como uma **caixa preta**. Não interessa como funciona, **o que importa é o resultado final**.
- O código de uma função deve ser o **mais independente possível** do resto do programa, e **genérico o suficiente** para ser reutilizado em outros projetos. (*Modularidade*).
- Uma função **pode ou não possuir parâmetros, pode ou não** retornar valor como resultado do seu trabalho.





# Procedimientos



# Procedimentos

A diferença entre um procedimento e uma função é que o primeiro não retorna valor ao contrário da função. Observe o exemplo abaixo:

```
1  #include <stdio.h>
2  void soma(int x, int y){
3      int r = x + y;
4      printf("a soma eh %d \n", r);
5  }
6
7  int main() {
8      soma(x: 10, y: 12);
9  }
```

**Neste exemplo os valores 10 e 12 são argumentos enviados a função que efetua o cálculo e ela mesma exibe o resultado na tela. Não devolve valor para a função que a chamou**



# Onde colocar suas funções



Em C, as funções podem ser posicionadas em qualquer parte do arquivo, **seja antes ou depois da função main(). No entanto**, existe **uma consideração importante a ser feita**:

- Se você optar por definir uma função após a função main(), é essencial fornecer ao compilador **um protótipo dessa função**. Este protótipo, também conhecido como declaração de função, informa ao compilador sobre a assinatura da função (ou seja, seu tipo de retorno e os tipos de seus parâmetros) antes de sua definição real. A declaração do protótipo é semelhante à definição da função, mas termina com um ponto e vírgula (;).





Inclua **os protótipos das funções no topo de seus programas**. Isso fornece ao compilador uma visão antecipada das funções que serão empregadas, permitindo que ele verifique se cada chamada de função está em conformidade com sua definição prevista.



**Passagem**



# Passagem de argumentos

A passagem de argumentos para funções em C pode ser feita de duas maneiras principais:

- por valor e
- por referência.



# Passagem por valor

- Neste método, o **valor da variável real (ou o valor real)** é **passado para a função**. Alterações feitas no valor do **parâmetro dentro da função não afetam a variável real no chamador**.



# Passagem por valor – Exemplo

```
#include <stdio.h>

void alteraPorValor(int x) {
    x = 50;
    printf("Valor dentro da função alteraPorValor: %d\n", x);
}

int main() {
    int a = 10;
    alteraPorValor(a);
    printf("Valor de a após chamar alteraPorValor: %d\n", a);
    return 0;
}
```

No exemplo acima, o valor de 'a' não é alterado, mesmo que o valor de x seja alterado na função alteraPorValor.



# Passagem por referência

- Neste método, o **endereço da variável real (ou a referência da variável)** é passado para a função. Isso significa que qualquer alteração **feita na variável referenciada dentro da função reflete na variável real do chamador.**
- Em C, a **passagem por referência é realizada passando** **ponteiros** como argumentos de função.

# Passagem por referência – Exemplo

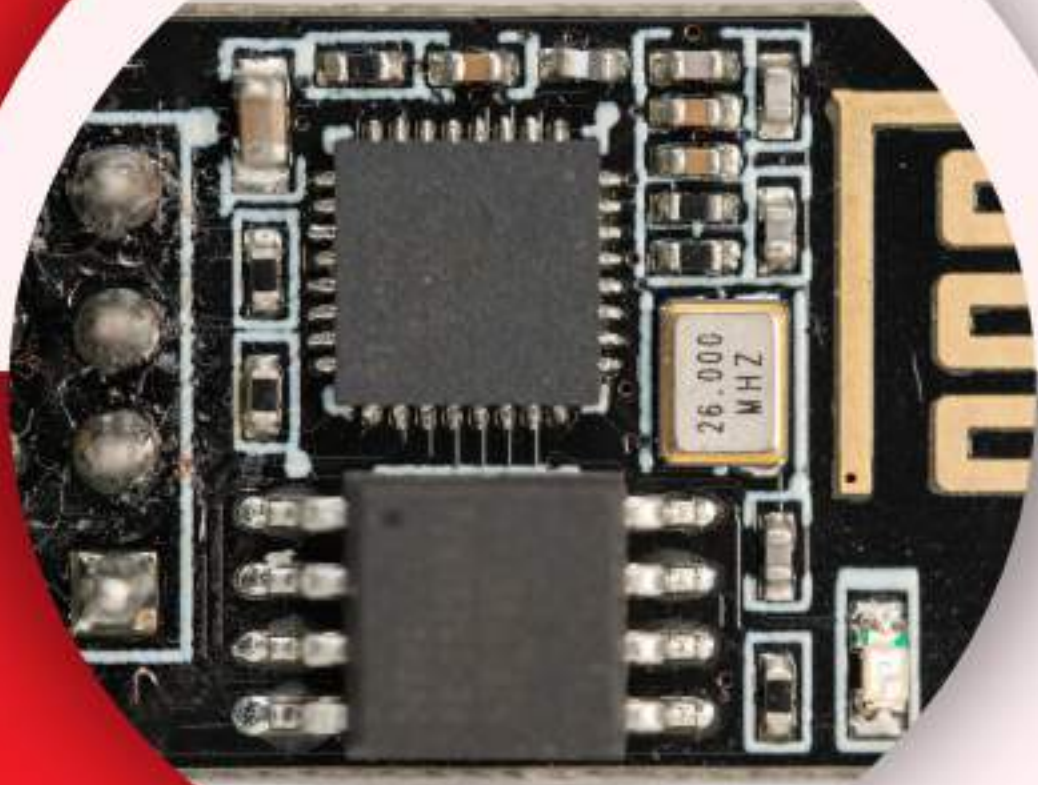
```
#include <stdio.h>

void alteraPorReferencia(int *x) {
    *x = 50;
    printf("Valor dentro da função alteraPorReferencia: %d\n", *x);
}

int main() {
    int a = 10;
    alteraPorReferencia(&a);
    printf("Valor de a após chamar alteraPorReferencia: %d\n", a);
    return 0;
}
```

No exemplo, o valor de **a** é alterado porque passamos o **endereço de a** para a função e, em seguida, **modificamos o valor no endereço referenciado**.





# Modularização

# Modularização

Modularizar programas em C é **uma prática fundamental para tornar o código mais organizado, reutilizável e fácil de manter**. A ideia é separar o código em **diferentes arquivos**, com base em sua funcionalidade, e depois "linkar" esses arquivos juntos para formar um programa executável.

Geralmente, essa separação é feita usando arquivos de código-fonte **(.c)** e arquivos de cabeçalho **(.h ou headers)**.



# Vantagens da Modularização

**Reutilização de Código:** Funções e estruturas comuns podem ser colocadas em arquivos separados e reutilizadas em diferentes programas.

**Facilidade de Manutenção:** Alterações em uma parte específica do programa (por exemplo, uma função) podem ser feitas sem ter que vasculhar um único arquivo de código grande.

**Colaboração:** Múltiplos desenvolvedores podem trabalhar em diferentes módulos sem interferir uns nos outros.

Imagine que temos um programa que faz operações matemáticas básicas. Vamos modularizá-lo.

Arquivos necessários:

- **math\_operations.h** (arquivo de cabeçalho)
- **math\_operations.c** (arquivo de código-fonte)
- **main.c** (arquivo principal)



# Arquivo: math\_operations.h

```
/* Documentação:  
 * Este arquivo contém declarações de funções para operações matemáticas b  
 */  
  
#ifndef MATH_OPERATIONS_H  
#define MATH_OPERATIONS_H  
  
// Protótipos de funções  
float add(float a, float b);  
float subtract(float a, float b);  
  
#endif // MATH_OPERATIONS_H
```

# Arquivo: math\_operations.c

```
#include "math_operations.h"

float add(float a, float b) {
    return a + b;
}

float subtract(float a, float b) {
    return a - b;
}
```



# Arquivo: main.c

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

int main() {
    float num1 = 5.5, num2 = 3.0;

    printf("Soma: %f\n", add(num1, num2));
    printf("Subtração: %f\n", subtract(num1, num2));

    return 0;
}
```

- Usamos diretivas de pré-processamento (**#ifndef, #define, #endif**) no **arquivo de cabeçalho** para evitar a **inclusão múltipla, o que pode causar erros de compilação**.
- O arquivo **math\_operations.c contém as definições das funções**, enquanto o arquivo **math\_operations.h contém apenas as declarações (protótipos)**. Isso permite que outros arquivos de código-fonte saibam sobre essas funções sem precisar saber exatamente como elas são implementadas.





# Exercícios - Aula 06

## Funções

### 1. Calculadora Básica:

- Crie funções para adição, subtração, multiplicação e divisão.
- O programa deve permitir ao usuário escolher uma operação e fornecer dois números. A operação correspondente deve ser realizada e o resultado impresso.

### 2. Conversão de Temperatura:

- Escreva uma função que converta temperaturas de Celsius para Fahrenheit e outra para converter de Fahrenheit para Celsius.

### 3. Manipulação de Strings:

- Crie uma função que receba uma string e retorne sua versão invertida.



# Exercícios - Aula 06

## Funções

### 4. Matriz de Transposição:

- Crie uma função que transponha uma matriz 3x3 (transforme linhas em colunas e vice-versa).

### 5. Maior e Menor em Vetor:

- Escreva uma função que aceite um vetor de números e retorne o maior e o menor número.

### 6. IMC:

- Baseado no exemplo anterior sobre IMC, expanda o programa para calcular o IMC de vários indivíduos, usando funções para ler dados, calcular IMC e imprimir resultados





# Exercícios - Aula 06

## Funções

### 7. String para Inteiro:

- Sem usar a função **atoi**, crie uma função que converta uma string numérica para um valor inteiro.

### 8. Sequência de Fibonacci:

- Crie uma função que gere os primeiros **n** números da sequência de Fibonacci em um vetor.

### 9. Classificação de Estudantes:

- Crie um programa que permita ao usuário inserir notas de estudantes em um vetor. Use funções para calcular a média, a nota mais alta e a nota mais baixa.

# Referências

---

DAMAS, L. M. D. Linguagem C. LTC, 2007.

---

HERBERT, S. C completo e total. 3a. ed. Pearson, 1997.

---

