

# **Algoritmos e Estruturas de Dados**

## ***(Aula 3 - Funções e Ponteiros em C/C++)***

Prof. Me. Diogo Tavares da Silva  
contato: [diogotavares@unibarretos.com.br](mailto:diogotavares@unibarretos.com.br)

# Funções

- Blocos de código que realizam uma tarefa específica.
  - “mini-programas” que são chamados dentro do programa principal.
- Tem o intuito de:
  - Ser reutilizáveis em diferentes partes do programa.
  - Organizar a estrutura do programa, facilitando a leitura e a manutenção do código.

# Funções

- Também são conhecidas como:
  - **rotinas**
  - **procedimentos (procedures)**
  - **métodos**
  - **subrotinas**
  - **subprogramas**

# Benefícios do uso de funções

- **Reutilização de código:** Evita a repetição de código e torna o programa mais eficiente.
- **Modularização:** Permite dividir o programa em partes menores e mais gerenciáveis.
- **Organização:** Facilita a leitura e a compreensão do código.
- **Manutenibilidade:** Facilita a correção de erros e a adição de novas funcionalidades.

# Benefícios do uso de funções

- **Reutilização de código:** Evita a repetição de código e torna o programa mais eficiente.
- **Modularização:** Permite dividir o programa em partes menores e mais gerenciáveis.
- **Organização:** Facilita a leitura e a compreensão do código.
- **Manutenibilidade:** Facilita a correção de erros e a adição de novas funcionalidades.

# Tipos de funções

- **Funções com retorno:** Retornam um valor ou referência para a função que a chamou.
- **Funções sem retorno:** Realizam alguma operação sem retornar nenhum valor ou referência ao fim da sua execução

# Tipos de funções

- **Funções embutidas (*built-ins*):** Disponibilizadas por bibliotecas “prontas” de utilidades de acompanham a linguagem.
- **Funções do usuário:** Funções que o próprio programador elabora e utilizada.

# Sintaxe de funções em C/C++

```
tipo_retorno nome_funcao( tipo_param1 nome_param1, tipo_param2 nome_param2, ...){  
    //código da função  
    ...  
    //  
    return valor //se houver retorno  
}
```

- **onde:**

- **tipo\_retorno:** tipo de dado a ser retornado pela função
- **nome\_função:** nome de função válido em C/C++
- **lista de parâmetros:** sempre descritos por tipo e nome do parâmetro
- **código da função:** onde a funcionalidade da função é programada
- **retorno:** retorna uma variável ou valor de retorno, se houver.



# Exemplo:

- **Área do círculo**

- **cálculo:**  $2 \cdot 3.1415 \cdot \text{raio}$
- **parâmetros de entrada:** valor do raio (decimal)
- **retorno:** a área do círculo (decimal)

# Exemplo:

- Área do círculo

```
#include <stdio.h>

double areaCirculo (double raio){
    return 2*3.1415*raio
}

main(){
    double r, area;
    puts("Digite o valor do raio do círculo: ");
    area = areaCirculo(r);
    printf("a área do círculo é %.3f",area);
}
```

# Escopo de variáveis

- Quando possuímos várias funções e blocos de código “{}” passamos a lidar com uma questão importante que é o escopo de variáveis
  - Região do código em que uma variável “é visível” (acessível)

# Escopo de variáveis

- **Variáveis locais:** São acessíveis apenas dentro do bloco “{...}” em que foram declaradas.
- **Variáveis globais:** São acessíveis em qualquer lugar do programa.

# Escopo de variáveis

- exemplo:

```
#include <stdio.h>

double areaGlobal = 0.0;

double areaCirculo (double raio){
    double area;
    area = 2*3.1415*raio
    return area
}

main(){
    double r, area;
    puts("Digite o valor do raio do círculo: ");
    area = areaCirculo(r);
    areaGlobal = areaGlobal + area;
    printf("a área do círculo é %.3f",area);
}
```

# Escopo de variáveis

- exemplo:

```
#include <stdio.h>

double areaGlobal = 0.0;

double areaCirculo (double raio){
    double area;
    area = 2*3.1415*raio;
    return area;
}

main(){
    double r, area;
    puts("Digite o valor do raio do círculo: ");
    area = areaCirculo(r);
    areaGlobal = areaGlobal + area;
    printf("a área do círculo é %.3f",area);
}
```

# Escopo de variáveis

- exemplo:

```
#include <stdio.h>

double areaGlobal = 0.0;

double areaCirculo (double raio){
    double area;
    area = 2*3.1415*raio
    return area
}

main(){
    double r, area;
    puts("Digite o valor do raio do círculo: ");
    area = areaCirculo(r);
    areaGlobal = areaGlobal + area;
    printf("a área do círculo é %.3f",area);
}
```



# Escopo de variáveis

- exemplo:

```
#include <stdio.h>

double areaGlobal = 0.0;

double areaCirculo (double raio){
    double area;
    area = 2*3.1415*raio
    return area
}

main(){
    double r, area;
    puts("Digite o valor do raio do círculo: ");
    area = areaCirculo(r);
    areaGlobal = areaGlobal + area;
    printf("a área do círculo é %.3f",area);
}
```



# Escopo de variáveis


- exemplo:

```
#include <stdio.h>

double areaGlobal = 0.0;

double areaCirculo (double raio){
    double area;
    area = 2*3.1415*raio
    return area
}

main(){
    double r, area;
    puts("Digite o valor do raio do círculo: ");
    area = areaCirculo(r);
    areaGlobal = areaGlobal + area;
    printf("a área do círculo é %.3f",area);
}
```



mesma variável?  
ou diferente?

# Escopo de variáveis

- **Observações:**

- Variáveis locais são mais seguras que as globais, pois evitam conflitos de nomes.
- Variáveis globais devem ser usadas com cuidado, pois podem ser facilmente modificadas por diferentes partes do programa.
  - Deixam o código dependente e de difícil manutenção

## Outro exemplo:

```
#include <stdio.h>

void atualizarValor(int valor){
    valor = 8
}
```

qual valor será impresso?

```
main(){
    int valor;
    valor = 4;
    atualizaValor(valor);
    printf("Valor: %d",valor);
}
```

**4 ou 8?**

# Variáveis do tipo Ponteiro (referência)

- Variáveis que armazenam o endereço de memória de outra variável.
  - Fazendo assim uma referência à uma lugar da memória
- Permitem acessar e modificar um valor de uma variável indiretamente.
- Em C/C++, são representados por um asterisco “\*” antes do nome da variável.
- Endereços por sua vez são representados por “&” antes do nome da variável

# Variáveis do tipo Ponteiro (referência)

- Exemplo:

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

```
int main(){
    setlocale(LC_ALL, "");
    int numero = 10;
    int *ponteiro;
```

```
    ponteiro = &numero; // Atribui o endereço de 'numero' a 'ponteiro'
    //cria a referencia entre ponteiro e número
```

```
    *ponteiro = 20;
```

```
    printf("Conteúdo da variável número: %d\n", numero);
    printf("Endereço de numero: %d\n", &numero);
    printf("Conteúdo da variável ponteiro: %d\n", ponteiro);
    printf("Conteúdo da variável que ponteiro faz referência: %d\n", *ponteiro);
}
```

# Variáveis do tipo Ponteiro (referência)

- Saída:

```
Conteúdo da variável número: 20  
Endereço de numero: 7994900  
Conteúdo da variável ponteiro: 7994900  
Conteúdo da variável que ponteiro faz referência: 20
```

# Tipos de passagem de parâmetros

- **Passagem por cópia:** Uma cópia do valor da variável é passada para a função.
  - Modificações na variável dentro da função não afetam a variável original.
- **Passagem por referência:** O endereço da variável é passado para a função por um parâmetro do tipo ponteiro (referência).
  - Modificações na variável dentro da função afetam a variável original.



# Passagem por cópia

```
#include <stdio.h>

void atualizarValor(int valor){
    valor = 8
}
```

qual valor será impresso?

```
main(){
    int valor;
    valor = 4;
    atualizaValor(valor);
    printf("Valor: %d",valor);
}
```

**4 ou 8?**



# Passagem por referência:

```
#include <stdio.h>

void atualizarValor(int *pValor){
    *pValor = 8
}
```

qual valor será impresso?

```
main(){
    int valor;
    valor = 4;
    atualizaValor(&valor);
    printf("Valor: %d", valor);
}
```

**4 ou 8?**

# Passagem de parâmetros

- Quando usar cada método:
  - **Passagem por cópia:**
    - Quando você deseja evitar que a função modifique a variável original.
    - Quando a variável é um tipo de dado simples (int, float, etc.).
  - **Passagem por referência:**
    - Quando você deseja que a função modifique a variável original.
    - Quando a variável é um tipo de dado complexo (vetor, matriz, registro, etc.)

# Próxima aula:

- Estruturas compostas homogêneas
  - Vetores
  - Matrizes

# Exercícios de fixação

- 1. Calculadora básica:
- Crie uma função para cada operação matemática básica (soma, subtração, multiplicação e divisão).
- A função deve receber dois números como parâmetros e retornar o resultado da operação.
- No main, chame cada função e imprima o resultado na tela.

# Exercícios de fixação

- 2. Maior e menor número:
- Crie uma função que recebe três números como parâmetros e retorna o maior e o menor entre eles.
- No main, chame a função e imprima o maior e o menor número na tela.

# Exercícios de fixação

## 3. Média de uma sequência de números:

- Crie uma função que recebe uma sequência de números como parâmetros e retorna a média aritmética dos números.
- A função deve ler os números do usuário até que ele digite um valor negativo.
- No main, chame a função e imprima a média na tela.

# Exercícios de fixação

## 4. Fatorial de um número:

- Crie uma função que recebe um número como parâmetro e retorna o seu fatorial.
- O fatorial de um número é o produto de todos os números inteiros positivos menores ou iguais a ele.
- No main, chame a função e imprima o fatorial do número na tela.

# Exercícios de fixação

## 5. Verificação de se um número é primo:

- Crie uma função que recebe um número como parâmetro e retorna true se ele for primo e false caso contrário.
- Um número primo é um número natural maior que 1 que possui apenas dois divisores distintos: 1 e ele próprio.
- No main, chame a função e informe se o número é primo ou não.



# Exercícios de fixação

## 6. Conversão de temperatura:

- Crie uma função que converte temperatura de Celsius para Fahrenheit e vice-versa.
- A função deve receber a temperatura e a escala original como parâmetros e retornar a temperatura na escala desejada.
- No main, chame a função para converter algumas temperaturas e imprima os resultados na tela.

# Exercícios de fixação

## 7. Jogo de adivinhação:

- Crie uma função que gera um número aleatório entre 1 e 100.
- O jogador deve tentar adivinhar o número, e a função deve fornecer dicas sobre se o palpite é maior ou menor que o número secreto.
- O jogo termina quando o jogador adivinhar o número secreto.