5950257 - Programação de Computadores

Aula 10 - Funções

Prof. Dr. Denis M. L. Martins
DCM | FFCLRP | USP

Objetivos de Aprendizagem

- Justificar a importância das funções para a manutenção, clareza e reutilização do código
- Identificar e utilizar corretamente os componentes de uma função em C
- Distinguir o fluxo de execução do programa principal (main) da execução interna das funções chamadas (cadeia de chamadas).
- Compreender o funcionamento da Call Stack na execução de funções.
- Entender o conceito de recursão e sua aplicação.

Prof. Dr. Denis M. L. Martins | martins.denis@usp.br

Motivação

Produza um código em C que imprima um número certo de estrelas (use a variável **int numOfStars**).

```
Para numOfStars = 1 -> *
Para numOfStars = 5 -> *****
```

Como você faria para produzir **essa mesma saída** com números diferentes de estrelas e em **partes diferentes** do código?

```c
#include <stdio.h>

int main() {
    printf("Imprimindo 5 estrela:\n");
    // Saída produzida pela primeira vez
    printf("Imprimindo 2:\n");
    // Saída produzida pela primeira vez
    return 0;
}
```


Prof. Dr. Denis M. L. Martins | martins.denis@usp.br

Função

Uma função é um bloco de código nomeado que executa uma tarefa específica e bem definida. Ela encapsula lógica, permitindo que o programa seja dividido em partes menores e gerenciáveis.

```
void printStars(int numOfStars){
    for (int star = 1 star <= numOfStars; star++){
        printf('*');
    }
    printf("\n");
}
```

> Note que temos utilizado várias funções em nossas aulas. **printf**, **scanf** e **sizeof** são funções!

Prof. Dr. Denis M. L. Martins | martins.denis@usp.br

Por que usar Funções? (Modularização)

1. **Reutilização de Código (DRY - Don't Repeat Yourself):** Escreva a lógica uma vez e chame-a quantas vezes for necessário.
2. **Manutenibilidade:** Se houver um bug, você sabe exatamente em qual "módulo" procurar.
3. **Clareza e Legibilidade:** O código principal (`main`) se torna uma sequência de chamadas lógicas, facilitando o entendimento do fluxo geral.

Prof. Dr. Denis M. L. Martins | martins.denis@usp.br

Estrutura de uma Função em C

1. *return* or *output parameter* type
2. function *name*
3. *type* of *input* paramters
4. *input* paramters
5. body of the function

```
①    ②         ③   ④
void printStars(int numOfStars) {
  for (int star = 1; star <= numOfStars; star++) {
    printf("%c", '*');
  }

  printf("\n");  // to start a newline
}
```

> Fonte da Imagem: Snefru: Learning Programming with C

Prof. Dr. Denis M. L. Martins | martins.denis@usp.br

Estrutura de uma Função em C (cont.)

Toda função possui:

1. **Tipo de Retorno:** O tipo de dado que a função promete devolver (ex: **int**, **float**, **void**). Se não retorna nada, usamos **void**.
2. **Nome da Função:** Identificador único para chamar o bloco de código.
3. **Parâmetros:** Os dados de entrada necessários para a função operar. São definidos entre parênteses (ex: **(int a, float b)**).
4. **Corpo da Função:** O conjunto de instruções que executam a lógica.

```
int calculaArea(float base, float altura) {
    return base * altura; // O corpo que executa o cálculo
}
```


Prof. Dr. Denis M. L. Martins | martins.denis@usp.br

Protótipo e Definição

Protótipos (Declaração): É o "mapa" do compilador. Deve ser declarado *antes* de qualquer função que o utilize em **main()** ou em outra função superior.

```
// Sintaxe: TipoRetorno NomeFuncao(TiposParametros);
int calculaArea(float base, float altura); // Protótipo
```

Definição: É a implementação real do código da função.

```
int calculaArea(float base, float altura) {
    return base * altura; // O corpo que executa o cálculo
}
```


Prof. Dr. Denis M. L. Martins | martins.denis@usp.br

Exemplo: Protótipo e Definição

```c
#include <stdio.h>
// ERRO: Chamada antes da declaração (se não houver protótipo)
printf("%d", somar(1, 2));

// CORRETO: Protótipo antes do uso
int somar(int a, int b);

int main() {
    printf("%d", somar(1, 2)); // Compila sem aviso
    return 0,
}

// Definição pode vir depois (se houver protótipo)
int somar(int a, int b) { return a + b; }
```


Prof. Dr. Denis M. L. Martins | martins.denis@usp.br

Fluxo de Execução e Retorno de Valores

Chamada: Para usar a função, você deve chamá-la passando os argumentos corretos. O compilador executa o código da função *pausando* o fluxo principal até que ela retorne um valor.

```
// Chamada na main:
float area = calculaArea(10.5, 4.0); // Passamos os valores 10.5 e 4.0
printf("A área é: %.2f\n", area);
```

Valor de Retorno (`return`): A palavra-chave **`return`** finaliza a execução da função e envia o valor especificado para quem chamou.

- Se você declarar **`void`**, nunca deve usar **`return`** com um valor (apenas **`return;`**).
- Se o tipo de retorno declarado não corresponder ao tipo de dado retornado, o compilador emitirá um erro.

Prof. Dr. Denis M. L. Martins | martins.denis@usp.br

Exercício: Calculadora

Crie um programa que pede ao usuário dois números **float primeiroNumero** e **float segundoNumero** e calcula as seguintes operações (usando funções separadas para cada uma delas):

- Soma
- Subtração
- Divisão (trate o caso de divisão por zero)
- Multiplicação

Prof. Dr. Denis M. L. Martins | martins.denis@usp.br

Passagem por Valor

A função recebe uma **cópia** do valor original da variável.

- **Consequência**: Qualquer alteração feita dentro da função afeta *apenas* a cópia local, e o original permanece intocado.

Exemplo:

```
void func(int x) {
    x = 50; // O valor original de x não muda.
}
```


Prof. Dr. Denis M. L. Martins | martins.denis@usp.br

Passagem por Referência

Em C, para simular passar por referência, passamos o **endereço de memória** da variável usando ponteiros (`*`).

- **Consequência**: A função pode acessar e modificar a variável original diretamente através do endereço.

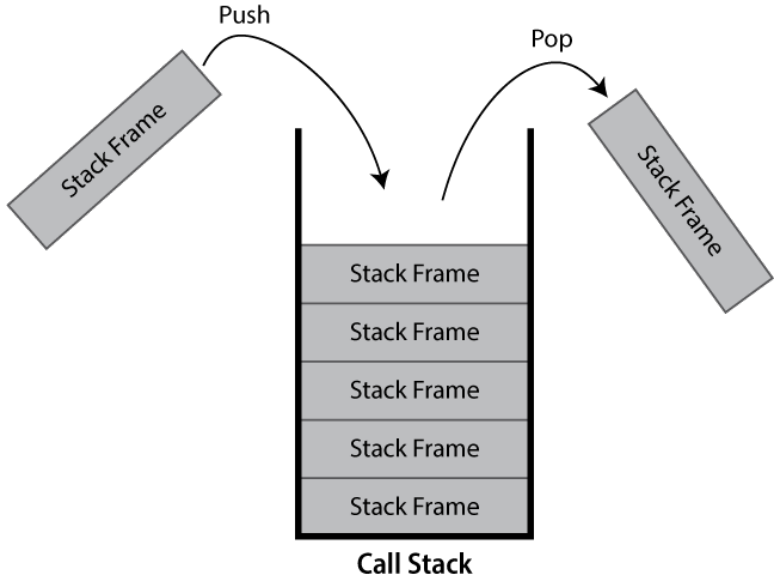
Exemplo:

```
void func(int *x) {
    *x = 50; // O valor original de x MUDARÁ.
}
```


Prof. Dr. Denis M. L. Martins | martins.denis@usp.br

Call Stack

- É uma estrutura de dados LIFO (Last-In, First-Out: Último a Entrar, Primeiro a Sair) usada pelo sistema operacional para gerenciar chamadas de função.
- Cada vez que uma função é chamada, um "quadro" ou frame é empilhado na Call Stack. Este quadro contém:
  - Parâmetros: Os valores passados para a função.
  - Variáveis Locais: As variáveis declaradas dentro da função.
  - Endereço de Retorno: O endereço exato no código onde o programa deve continuar após terminar a execução desta função específica.



> Fonte da imagem: University of Sheffield

Prof. Dr. Denis M. L. Martins | martins.denis@usp.br

Exemplo

```c
 #include <stdio.h>

void funcaoA() {
    // Passo 3: Executa Bloco de código de A
    printf("--- funcaoA ---\n");
}

int main() {
    // Passo 1: Início do programa principal.
    printf("[MAIN] Início do programa principal.\n");
    // Passo 2: Chama A. O fluxo para aqui até que A retorne.
    funcaoA();
    // Passo 4: Executado APENAS após o retorno de A.
    printf("[MAIN] Fim da execução principal.\n");
    return 0;
}
```


Prof. Dr. Denis M. L. Martins | martins.denis@usp.br

Exercício: Analise a sequência de chamadas

```c
#include <stdio.h>

void funcaoC() {
    printf("C ");
}
void funcaoB() {
    printf("B1 ");
    funcaoC();
    printf("B2 ");
}
void funcaoA() {
    printf("A1 ");
    funcaoB();
    printf("A2 ");
}
int main() {
    printf("M1 ");
    funcaoA();
    printf("M2 ");
    return 0;
}
```


Prof. Dr. Denis M. L. Martins | martins.denis@usp.br

Recursividade

Recursão é quando uma função chama a si mesma para resolver um problema.

Componentes:

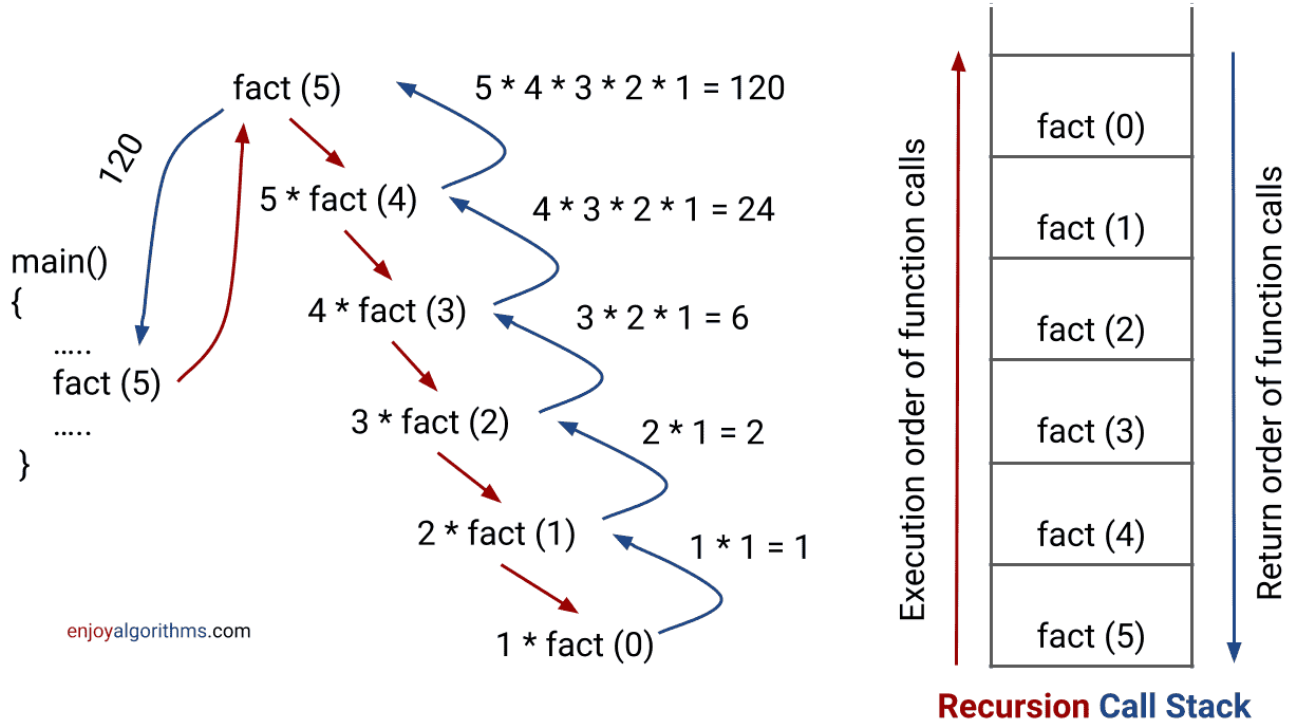
- **Caso Base (Base Case):** A condição de parada obrigatória. Sem ela, o programa entrará em *Stack Overflow* (estouro da pilha).
- **Passo Recursivo (Recursive Step):** O passo onde a função se chama novamente, mas com uma entrada que move o problema progressivamente para o Caso Base.

Exemplo:

```
int fatorial(int n) {
    if (n == 0 || n == 1) { // CASO BASE
        return 1;
    } else {
        return n * fatorial(n - 1); // PASSO RECURSIVO
    }
}
```


Prof. Dr. Denis M. L. Martins | martins.denis@usp.br

Passo-a-passo da execução da função fatorial



> A Call Stack armazena os resultados parciais e garante que, quando o caso base é resolvido, os resultados sejam propagados "para cima" (pop) até a chamada inicial. Fonte da imagem: Enjoy Algorithms.

Prof. Dr. Denis M. L. Martins | martins.denis@usp.br

Exercício: Fibonacci

Crie uma função chamada **fibonacci** que recebe um número inteiro **n** como parâmetro e retorna os **n** primeiros termos da sequência de Fibonacci.

Prof. Dr. Denis M. L. Martins | martins.denis@usp.br

Exercício: Soma de elementos de um vetor

Escreva uma função recursiva **soma_vetor(int vetor[], int tamanho)** que calcule a soma dos elementos de um array, sem usar laços (**for** ou **while**).

Prof. Dr. Denis M. L. Martins | martins.denis@usp.br

Limitação de Memória

Stack Overflow: É o erro mais comum em recursão. Acontece quando a Call Stack excede o espaço de memória alocado para ela (geralmente por causa da falta ou falha do Caso Base). Gera um segmentation fault ou mensagem específica de stack overflow.

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

void function(void) {
    function();
}

int main(void) {
    function();
    return 0;
}
```


Prof. Dr. Denis M. L. Martins | martins.denis@usp.br

Criando e Usando Bibliotecas

- **Separação de Responsabilidade:** Quem usa a função (cliente) não precisa saber como ela é implementada.
- **Reutilização:** O mesmo código pode ser usado em múltiplos projetos sem duplicação.

> Vamos utilizar a IDE https://www.onlinegdb.com/online_c_compiler para criação e utilização de bibliotecas em C.

Prof. Dr. Denis M. L. Martins | martins.denis@usp.br

Arquivo de Cabeçalho (.h)

- **Apenas Declarações:** O `.h` deve conter apenas protótipos de funções, definições de constantes e estruturas.
- **Nenhuma Implementação (Geralmente):** Não coloque `{ ... }` dentro do `.h`. Isso gera erro de múltiplas definições se incluído em vários arquivos.

```c
// mathutils.h
#ifndef MATHUTILS        // 1. Proteção contra inclusão múltipla
#define MATHUTILS        // 2. Define o nome do arquivo

#include <stdio.h>          // 3. Inclui bibliotecas padrão (opcional)

int somar(int a, int b);    // 4. Protótipo da função
void imprimirResultado(int x);

#endif                      // 5. Fim da proteção
```


Prof. Dr. Denis M. L. Martins | martins.denis@usp.br

Arquivo de Fonte (.c): Source Code

- **Apenas Definições:** O `.c` contém a implementação real das funções declaradas no `.h`.
- **Inclusão do Header:** O arquivo `.c` deve incluir seu próprio `.h` para garantir que as variáveis globais ou ponteiros estejam corretos.

```c
// mathutils.c
#include "mathutils.h"     // Inclui o cabeçalho (para ver protótipos)

int somar(int a, int b) {   // Implementação real
    return a + b;
}

void imprimirResultado(int x) {
    printf("Resultado: %d\n", x);
}
```

> Se o `.c` não incluir o `.h`, o compilador pode não saber os tipos dos parâmetro.

Prof. Dr. Denis M. L. Martins | martins.denis@usp.br

Arquivo Principal (Cliente)

```c
#include <stdio.h>
#include "mathutils.h"

int main()
{
    int resultado = somar(2, 3);
    imprimirResultado(resultado);

    return 0;
}
```


Prof. Dr. Denis M. L. Martins | martins.denis@usp.br

Compilação e Linking

Para criar um executável a partir de uma biblioteca local, o processo tem duas etapas:

1. **Compilação (`.c` -> `.o`):** Transforma código fonte em objeto.

```
gcc -c mathutils.c        # Gera mathutils.o
gcc -c main.c              # Gera main.o
```

2. **Linking (`.o` -> Executável):** Junta os objetos e resolve endereços.

```
gcc main.o mathutils.o -o programa
```


Prof. Dr. Denis M. L. Martins | martins.denis@usp.br

Dúvidas e Discussão

Prof. Dr. Denis M. L. Martins | martins.denis@usp.br

Exercício e Questões

Questões teóricas

1. Por que é importante adicionar protótipos de função no início de um programa?
2. Considere as seguintes funções e variáveis globais **A = 10**.

```c
void alterar_valor(int x) { x = 5; } // Função A
void alterar_por_referencia(int *p) { *p = 20; } // Função B

// Código principal:
alterar_valor(A);
alterar_por_referencia(&A);
```

Qual será o valor final de **A** e por quê? Explique detalhadamente o mecanismo de passagem em cada função.

Prof. Dr. Denis M. L. Martins | martins.denis@usp.br

Exercício 1

Crie uma função chamada **calcular_media** que receba três notas (**float n1**, **float n2**, **float n3**) como parâmetros e retorne a média aritmética dessas notas. Use esta função no seu programa principal para calcular e imprimir a média de um aluno.

Depois que estiver com o código funcionando, modifique a função para receber um ponteiro de notas e retornar a média aritmética dessas notas. As notas devem ser lidas do teclado na função **main()**.

Prof. Dr. Denis M. L. Martins | martins.denis@usp.br

Exercício 2

Escreva uma função chamada **inverterNumeros** que receba dois ponteiros inteiros **int *a** e **int *b**. Esta função deve trocar os valores apontados pelos ponteiros, **sem usar variáveis temporárias** na função.

Prof. Dr. Denis M. L. Martins | martins.denis@usp.br

Exercício 3

Escreva uma função recursiva **int pot(int base, int expoente)** que calcule:

$$base^{expoente}$$

> **Dica:** Lembre-se de como o caso base deve ser tratado quando o expoente é 0 ou 1.

Prof. Dr. Denis M. L. Martins | martins.denis@usp.br

Exercício 4

Crie uma biblioteca simples de matemática vetorial para calcular a distância euclidiana entre dois pontos 2D.

- Crie o arquivo **pontos.h**. Ele deve conter um protótipo de função chamada **calcular_distancia** que recebe duas coordenadas **x1**, **y1**, e **x2**, **y2**.
- Crie o arquivo **pontos.c**. Ele deve:
  - Incluir **"pontos.h"**.
  - Implementar a função: **calcularDistanciaEuclidiana(double x1, double y1, double x2, double y2)**.
  - A fórmula da distância Euclidiana é $\sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$.
- Crie o arquivo **main.c**. Ele deve:
  - Incluir **"pontos.h"**.
  - Declarar um ponto inicial **x1**, **y1** e um ponto final **x2**, **y2**.
  - Chamar a função **calcularDistanciaEuclidiana** com as coordenadas de dos pontos inicial e final.
  - Imprimir o resultado formatado (ex: "Distância entre A e B é X.XX").

Prof. Dr. Denis M. L. Martins | martins.denis@usp.br