Linguagem de Programação I

Aula 8 – Funções e Recursividade

Prof Ana Carolina Sokolonski

Bacharelado em Sistemas de Informação
Instituto Federal da Bahia – Campus Feira de Santana

2026

Agenda

Funções

Modularização em C

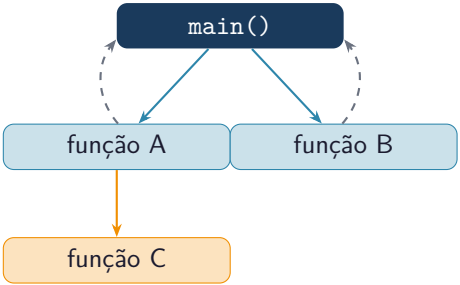
O que é uma Função?

Definição

Uma **função** é uma sub-rotina que realiza uma **tarefa específica** em um módulo de código, referenciada pelo programa principal através de um nome.

Por que usar funções?

- **Modularização**: dividir o problema em partes
- **Reuso**: chamar o mesmo código várias vezes
- **Legibilidade**: código mais organizado
- **Manutenção**: alterar em um só lugar



Funções chamam outras funções e retornam ao chamador.

Estrutura de uma Função

Sintaxe geral

```
tipo_retorno nome(parametros) {
  instrucoes;
  return valor;
}
```

Componentes

- **tipo_retorno**: tipo do valor devolvido `int`, `float`, `void`...
- **nome**: identificador da função
- **parâmetros**: variáveis de entrada
- **return**: devolve o resultado

Passagem de Parâmetros

Os **parâmetros** são variáveis declaradas no cabeçalho da função. Permitem a **comunicação** entre a função e o programa principal — chamamos de **passagem de valores**.

Exemplo

```
int multiplica(int n1, int n2) {
  return n1 * n2;
}
```

Exemplo: Função `multiplica()`

```c
#include <stdio.h>

int multiplica(int N1, int N2) //multiplica recebe N1,N2 e retorna um int
{
  int resultado;
  resultado = N1 * N2;
  return(resultado); //retornando o valor para main
}
/****************** função principal (main) ********************/
void main(void){
  int V1, V2, resultado;
  printf("Digite o primeiro valor:");
  scanf("%d", &V1);
  printf("Digite o segundo valor:");
  scanf("%d", &V2);
  resultado = multiplica(V1,V2);

  printf("Resultado = %d\n", resultado);
}
```

Protótipo de Função

Declarando a interface

Protótipo de Função

O que é?

O protótipo é uma **declaração antecipada** da interface da função, colocado antes da `main()`. Informa ao compilador:

- O **tipo de retorno**
- O **nome** da função
- A **lista de parâmetros**

Por que usar?

Permite definir a função *após* a `main()`, mantendo o código mais organizado. Sem o protótipo, a função deve aparecer *antes* do seu uso.

Sintaxe

```
// protótipo (antes da main)
int multiplica(int n1, int n2);

int main() {
  int r = multiplica(3, 4);
}

// definição (depois da main)
int multiplica(int n1, int n2) {
  return n1 * n2;
}
```

Exemplo: Protótipo da Função `multiplica()`

```c
#include <stdio.h>

int multiplica(int N1, int N2); //protótipo da função multiplica
/****************** função principal (main) ********************/
void main(void){
  int V1, V2, resultado;
  printf("Digite o primeiro valor:");
  scanf("%d", &V1);
  printf("Digite o segundo valor:");
  scanf("%d", &V2);
  resultado = multiplica(V1,V2);

  printf("Resultado = %d\n", resultado);
}

int multiplica(int N1, int N2) //multiplica recebe N1,N2 e retorna um int
{
  int resultado;
  resultado = N1 * N2;
  return(resultado); //retornando o valor para main
}
```

Função sem Retorno

`void` — quando não há valor a devolver

Função `void` — Sem Retorno

Quando usar `void`?

Quando a função executa um bloco de comandos mas **não precisa retornar nenhum valor** — por exemplo, funções que apenas exibem resultados na tela.

Sintaxe

```
void nome(void) {
  instrucoes;
  // sem return
}
```

`void` nos parâmetros

Se a função **não recebe parâmetros**, também usamos `void` no lugar da lista de parâmetros.

```
1  #include <stdio.h>
2
3  void imprime_cabec(void){
4    printf("*****************************\n");
5    printf("*       LINGUAGEM C         *\n");
6    printf("*****************************\n");
7
8    return; /* retorno de uma função void  */
9  }
10
11 void main(){
12     imprime_cabec();
13 }
```

Recursividade

Funções que chamam a si mesmas

O que é Recursividade?

Estrutura recursiva

Muitos problemas têm a propriedade de que **cada instância contém uma instância menor do mesmo problema**. Dizemos que têm **estrutura recursiva**.

Método

1. Se a instância for **pequena** → resolva diretamente (**caso base**)
2. Senão → **reduza** e aplique o mesmo método (**passo recursivo**)

Exemplo: Fatorial

```
fat(0) = 1 // caso base
fat(n) = n * fat(n-1)
// passo recursivo
```

$5! = 5 \times 4 \times 3 \times 2 \times 1 = 120$

Atenção

Toda função recursiva **deve** ter um caso base, caso contrário entra em loop infinito!

Exercícios

1. Faça um algoritmo recursivo que encontre o **maior valor** de um vetor.
2. Faça um algoritmo recursivo que calcule o **fatorial** de um número e mostre suas parcelas.
   Ex: $6! = 6 \times 5 \times 4 \times 3 \times 2 \times 1 = 720$
3. A **sequência de Fibonacci** é definida como:

   $F(0) = 0, \quad F(1) = 1, \quad F(n) = F(n-1) + F(n-2)$ para $n > 1$

   Implemente em C uma versão **recursiva** e uma **iterativa**.

Ex. 1 — Maior valor recursivo (parte 1)

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

int maior_r (int tam, int v[]){
    int maior;
    if (tam == 0) return -1;
    if (tam == 1) return v[0];
    maior = maior_r(tam-1,v);
    if (v[tam] > maior)
        return v[tam];
    else
        return maior;
}
```

Ex. 1 — Maior valor recursivo (parte 2)

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

int maior_r (int tam, int v[]){
    int maior;
    if (tam == 0) return -1;
    if (tam == 1) return v[0];
    maior = maior_r(tam-1,v);
    if (v[tam-1] > maior)
        return v[tam-1];
    else
        return maior;
}

int main (void){
int v1[]={134,3,234,7,567,5,678,2,899,0};
int v[100];
int tam, d=0;

for(tam=0;d!=-1;tam++){
    printf("Digite os elementos do vetor, digite -1 para sair\n");
    scanf("%d",&d);
    if(d!=-1)
     v[tam] = d;
}

  int i = maior_r(tam,v);
  printf("Maior item do Vetor= %d\n", i);
}
```

Ex. 2 — Fatorial Recursivo

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

int fat (int n){
    if (n == 1){
        printf("1 = ");
        return 1;
    }
    else{
        printf("%d*",n);
        return (n * fat(n-1));
    }
}

int main (void){
int n;

  printf("Informe o número que você deseja saber o fatorial:");
  scanf("%d",&n);
  printf("%d! = ",n);
  printf("%d\n", fat(n));
}
```

Ex. 3 — Fibonacci Iterativo

```c
void fibonacci (int n){
    int fib, fib1, fib2, i;

    if (n == 0) {
        printf("0\n");
    } else if (n == 1){
        printf("0, 1\n");
    } else{
        printf("0, 1, ");
        fib1 = 1;
        fib2 = 0;
        i = 2;
        while (i <= n){
            fib = fib1+fib2;
            if (i==n) printf("%d\n", fib);
            else printf("%d, ", fib);
            fib2 = fib1;
            fib1 = fib;
            i++;
        }
    }

}
```

Ex. 3 — Fibonacci Recursivo

```c
/**A função de Fibonacci é definida assim:   F(0) = 0,   F(1) = 1   e   F(n) = F(n-1) + F(n-2)
para n > 1. Descreva a função F em linguagem C. Faça uma versão recursiva e uma iterativa.**/
int fibonacci (int n){
    if (n == 0) return 0;
    else if (n == 1) return 1;
    else return fibonacci(n-1) + fibonacci(n-2);
}
int main (void){
int n, fib, i;

  printf("Informe o número de fatores das sequências de Fibonacci:");
  scanf("%d",&n);
  for (i = 0; i <= n; i++)
     printf("Parcela %d da sequência = %d\n",i,fibonacci(i));
}
```

FIM