Marisangila Alves, MSc
marisangila.alves@udesc.com
marisangila.com.br

UDESC
UNIVERSIDADE DO ESTADO DE SANTA CATARINA
JOINVILLE
CENTRO DE CIÊNCIAS TECNOLÓGICAS

UDESC
Universidade do Estado de Santa Catarina

2025/1

Algoritmo e Linguagem de Programação

Linguagem de Programação C
Função

Sumário

O que é?

Definição I

> *A maioria dos programas de computador que resolvem problemas do mundo real é muito maior do que os programas apresentados [...]. A experiência tem mostrado que a melhor maneira de desenvolver e manter um programa grande é construí-lo a partir de pequenas partes ou módulos, sendo cada uma delas mais fácil de manipular do que o programa original. Essa técnica é chamada dividir e conquistar.*
>
> (Deitel; Deitel, 2011)

- Uma função em lógica de programação é um bloco de código que realiza uma tarefa específica.
- Ela recebe entradas, processa essas entradas e, pode ou não, retornar um resultado.
- Funções são usadas para organizar e reutilizar código, tornando os programas mais eficientes e legíveis.
- Existem funções que podem ser escritas pelo programador e existem funções prontas, disponíveis na biblioteca padrão do C.

Funções conhecidas

- `printf()`
- `scanf()`
- `pow()`
- `rand()`

Por que?

Porque usar funções?

- `abs()` é uma função da biblioteca padrão e, tem como objetivo a partir de um número retornar o valor absoluto desse número.
- Portanto, o valor absoluto é obtido em apenas uma única linha, sem que o programador implemente a lógica para encontrar o valor absoluto de um número.

```c
#include <stdio.h>
#include <stdlib.h>
int main(){
    int absoluto = abs(-10);
    printf("%d", absoluto);
    return 0;
}
```

Código 1: Função valor absoluto `abs()`.

- Mas, e se a função abs() não existisse?

```c
#include <stdio.h>
int meu_abs(int numero) {
    if (numero < 0) {
        return -numero;
    } else {
        return numero;
    }
}
int main(){
    int absoluto = meu_abs(-10);
    printf("%d", absoluto);
    return 0;
}

```

Código 2: Implementação função valor absoluto.

Como saber as funções da biblioteca padrão fazem e como usá-las.

Consultando a documentação!

Biblioteca	Descrição
`stdio.h`	Fornece funções para entrada e saída padrão, como `printf()`, `scanf()` e manipulação de arquivos.
`stdlib.h`	Inclui funções para alocação de memória dinâmica, controle de processos, conversões e gerenciamento de ambiente, como `malloc()`, `free()` e `atoi()`.
`string.h`	Contém funções para manipulação de strings, como `strlen()`, `strcpy()`, `strcat()` e `strcmp()`.
`math.h`	Oferece funções matemáticas como `sqrt()`, `pow()`, `sin()`, `cos()` e `log()`.
`time.h`	Fornece funções para manipulação de data e hora, como `time()`, `clock()` e `difftime()`.

Tabela 1: Principais bibliotecas da linguagem C

O mundo sem printf()

Como seria se não existisse o `printf`?

```c
#include <stdio.h>
#include <stdarg.h>
int main() {
    my_printf("O inteiro eh %d e o float eh %f\n", 42, 3.14159);
    return 0;
}
```

Código 3: Chamada da função `printf()` implementada.

```c
void my_printf(const char* format, ...) {
    va_list args;
    va_start(args, format);
    while (*format) {
        if (*format == '%') {
            format++;
            if (*format == 'd') {
                int i = va_arg(args, int);
                printf("%d", i);
            } else if (*format == 'f') {
                double f = va_arg(args, double);
                printf("%f", f);
            }
        } else {
            putchar(*format);
        }
        format++;
    }
    va_end(args);
}
```

Código 4: Exemplo de implementação de função `printf()`.

É possível acessar o código fonte da função `printf()`.

Nota:

Ao utilizar uma função, não importa sua implementação. Apenas, quais parâmetros devem ser passados para que ela funcione.

Atenção!

O uso de funções amplamente utilizadas pode contribuir para o desempenho, uma vez que essas funções passaram por anos de refinamento. **Não reinvente a roda!**

Declarar

Criando novas funções

O que é preciso para definir uma função:

1. Uma função deve ter um nome:
   - O nome da função deve refletir seu objetivo;
   - O objetivo da função deve ser bem definido.
2. Uma função pode ter parâmentros:
   - Um parâmetro pode ser entendido como o que a função precisa para cumprir seu objetivo;
   - Um parâmetro é uma varíavel;
   - Pode ser necessário nenhum , um ou múltiplos parâmetros.
3. Uma Função pode ter um retorno;
   - O retorno é como o resultado da função;
   - O retorno não obrigatório.
4. Uma função tem um corpo:
   - O corpo de uma função é sua implementação.

5. Uma função tem uma assinatura.
6. Uma função só é executada quando chamada por outra função.

Exemplo:

- O nome da função `pow()` é pow!
- Para utilizar a função é necessário conhecer a assinatura da função.
- A assinatura deve ser declarada, assim como variáveis são declaradas!
- Sua assinatura é a seguinte:

```
1  int pow(int base, int expoente);
```

Atenção!

Nome de uma função é um identificador único que representa a função e é usado para chamá-la em outros lugares no programa.

Vamos analisar a assinatura da função por partes!

Declarar e Definir

```c
#include <stdio.h>
void somar(int a, int b);
int main()
{
    return 0;
}
void somar(int a, int b){
    int resultado = a + b;
    printf("%d", resultado);
}
```

Código 5: Exemplo função somar declarada e depois definida.

```c
#include <stdio.h>
void somar(int a, int b){
    int resultado = a + b;
    printf("%d", resultado);
}
void main()
{
    return 0;
}
```

Código 6: Exemplo função somar declarada e definida.

Parâmetros

Relembrando:

- São variáveis que a função recebe como entrada.
- Eles são opcionais, dependendo da função, e podem ser usados para passar informações necessárias para a função realizar sua tarefa.
- Uma função pode ter nenhum ou mais parâmetros.

- Sem parâmetro:

```
1        exibir_mensagem();
```

- Com parâmetro:

```
1        somar(int a, int b);
```

Chamada de função

- É o ato de invocar a função a partir de outra parte do programa.
- Isso é feito pelo nome da função, seguido por parênteses que podem conter argumentos (valores a serem passados como parâmetros).
- Parâmetro é do ponto de vista de implementação.
- Argumento é do ponto de vista da chamada.

- Chamada de uma função com a tarefa de somar dois números que deve receber dois argumentos como parâmetro.

```
1        somar(2,5);
```

- Chamada de função com a tarefa de exibir uma mensagem na tela. Um argumento deve ser passado.

```
1        exibir_mensagem("Bom dia");
```

Retorno

- Algumas funções retornam um valor como resultado de sua execução.
- Esse valor é chamado de valor de retorno e é devolvido à parte do programa que chamou a função.
- O valor de retorno deve ser armazenado em uma variável.
- O valor de retorno é opcional e pode ser de qualquer tipo de dado.

Agora, a função `somar()` não exibe o resultado, mas usa a instrução `return` para devolver o desultado para a função que a chamou.

```
1 int somar(int a, int b){
2     int resultado = a + b;
3     return resultado;
4 }
```

Código 7: Exemplo função somar com retorno.

- função que tem como tarefa de somar dois números e retornar o resultado. Dois argumentos devem ser passados.

```
1                resultado = somar(2,5);
```

- Função sem retorno com argumento:

```
1                exibir_mensagem("Bom dia");
```

```c
#include <stdio.h>
int somar(int a, int b){
    int resultado = a + b;
    return resultado;
}
int main()
{
    int x,y, soma;
    printf("Digite um numero:\n");
    scanf("%d", &x);
    printf("Digite outro numero:\n");
    scanf("%d", &y);
    soma = somar(x,y);
    printf("Resultado: %d", soma);
    return 0;
}
```

Código 8: Exemplo função somar.

Corpo da Função

- O corpo da função pode conter declarações de variáveis locais, estruturas de controle (como condicionais e laços), operações matemáticas e outras operações relevantes para a tarefa da função.

```c
int somar(int a, int b){
    int resultado = a + b;
    return resultado;
}
```

Código 9: Exemplo função somar.

Nota:

Uma função pode ser chamada por outra função!

Hierarquia

```c
#include <stdio.h>
#include <math.h>
double calcular(double numero){
    double potencia =  pow(numero, 2);
    double raiz = sqrt(potencia);
    return raiz;
}
int main()
{
    double numero,resultado;
    scanf("%lf",  &numero);
    resultado = calcular(numero);
    printf("%.2f", resultado);
    return 0;
}
```

Código 10: Função `main()`.

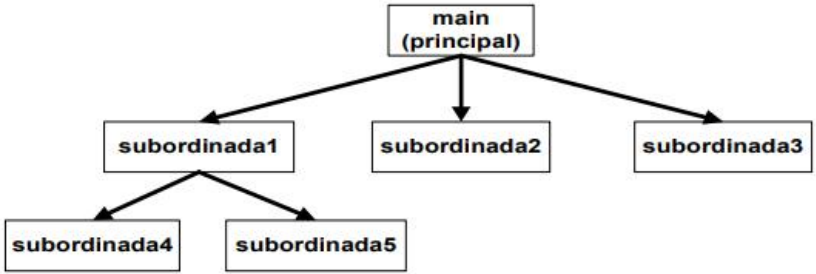

Figura 1: Relacionamento hierárquico de funções (Deitel; Deitel, 2011).

Escopo

- **Escopo Limitado**: As variáveis locais são declaradas dentro de uma função ou bloco específico do programa.
- **Acesso Restrito**: Elas só podem ser acessadas e usadas dentro desse escopo onde foram declaradas.
- **Vida Curta**: Geralmente, sua vida útil está limitada ao tempo de execução da função ou do bloco onde foram definidas.

- **Escopo Amplo**: As variáveis globais são declaradas fora de funções e blocos, tornando-as acessíveis em todo o programa.
- **Acesso Geral**: Elas podem ser acessadas e utilizadas em qualquer parte do código, em qualquer função ou bloco.
- **Vida Longa**: Sua vida útil persiste durante toda a execução do programa.

```c
#include <stdio.h>

char frase[50] = "esta uma variavel global";

void mostrar_mensagem(){
    printf("Dentro da Funcao: %s\n", frase);
}
int main()
{
    printf("Fora da Funcao: %s\n", frase);
    mostrar_mensagem();
    return 0;
}
```

Código 11: Exemplo função somar.

Otimização

```c
#include <stdio.h>
int verificar_par(int numero){
    int eh_par;
    if(numero % 2 == 0)
    {
        eh_par = 0; // VERDADEIRO
    }else{
        eh_par = 1; // FALSO
    }
    return eh_par;
}
int main()
{
    int n, validacao;
    printf("Digite um numero:\n");
    scanf("%d", &n);
    validacao = verificar_par(n);
    if (validacao)
    {
        printf("Eh par!\n");
    }else{
        printf("Eh impar!\n");
    }
    return 0;
}
```

Código 12: Exemplo função par.

```c
#include <stdio.h>
int verificar_par(int numero){
    return numero % 2 == 0;
}
int main()
{
    int n, validacao;
    printf("Digite um numero:\n");
    scanf("%d", &n);
    validacao = verificar_par(n);
    if (validacao)
    {
        printf("Eh par!\n");
    }else{
        printf("Eh impar!\n");
    }
    return 0;
}
```

Código 13: Exemplo função par otimizada.

Modularização

- **Reutilização de Código**: As funções permitem escrever uma lógica uma vez e reutilizá-la em várias partes do programa. Isso economiza tempo e evita duplicação de código.
- **Organização**: Funções ajudam a organizar o código, dividindo-o em módulos lógicos. Isso torna o código mais claro e fácil de entender.
- **Depuração**: Com funções menores e bem definidas, é mais fácil isolar e corrigir erros, pois você pode testar cada função individualmente.
- **Abstração**: As funções permitem abstrair detalhes de implementação. Você pode usar uma função sem se preocupar com como ela funciona internamente.
- **Colaboração**: Em projetos de equipe, funções permitem que diferentes membros trabalhem em partes separadas do código de forma independente.

(Deitel; Deitel, 2011) - Capítulo/Seção 5.5.

DE OLIVEIRA, J.F.; MANZANO, J.A.N.G. **Algoritmos: Lógica para Desenvolvimento de Programação de Computadores**. 16. ed. São Paulo: Editora Érica, 2004.

DE SOUZA, M.A.F. *et al.* **Algoritmos e Lógica de Programação**. São Paulo: Thomson Learning, 2004.

DEITEL, Paul; DEITEL, Harvey. **C: Como Programar**. 6. ed. São Paulo: Pearson Universidades, 2011.

MEDINA, M.; FERTIG, C. **Algoritmos e Programação – Teoria e Prática**. São Paulo: Novatec, 2005.

Estes slides estão protegidos por uma licença Creative Commons

Este modelo foi adaptado de Maxime Chupin.

Marisangila Alves, MSc
marisangila.alves@udesc.com
marisangila.com.br

UDESC
UNIVERSIDADE DO ESTADO DE SANTA CATARINA
JOINVILLE
CENTRO DE CIÊNCIAS TECNOLÓGICAS

UDESC
Universidade do Estado de Santa Catarina

2025/1

Algoritmo e Linguagem de Programação

Linguagem de Programação C
Função