

Fundamentos de Programação

Linguagem de Programação C
Função

Sumário

- | | | | |
|---|--------------------|----|-----------------|
| 1 | O que é? | 7 | Retorno |
| 2 | Por que? | 8 | Corpo da Função |
| 3 | Declarar | 9 | Hierarquia |
| 4 | Declarar e Definir | 10 | Escopo |
| 5 | Parâmetros | 11 | Otimização |
| 6 | Chamada de função | 12 | Modularização |

O que é?

*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.

```
1 #include <stdio.h>
2 #include <stdlib.h>
3 int main(){
4     int absoluto = abs(-10);
5     printf("%d", absoluto);
6     return 0;
7 }
```

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

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

```
1 #include <stdio.h>
2 int meu_abs(int numero) {
3     if (numero < 0) {
4         return -numero;
5     } else {
6         return numero;
7     }
8 }
9 int main(){
10     int absoluto = meu_abs(-10);
11     printf("%d", absoluto);
12     return 0;
13 }
14
```

Código 2: Implementação função valor absoluto.

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

 Consultando a documentação!

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

Tabela 1: Principais bibliotecas da linguagem C

O mundo sem printf()

Como seria se não existisse o `printf`?

```
1 #include <stdio.h>
2 #include <stdarg.h>
3 int main() {
4     my_printf("0 inteiro eh %d e o float eh %f\n", 42, 3.14159);
5     return 0;
6 }
```

Código 3: Chamada da função printf() implementada.

```

1 void my_printf(const char* format, ...) {
2     va_list args;
3     va_start(args, format);
4     while (*format) {
5         if (*format == '%') {
6             format++;
7             if (*format == 'd') {
8                 int i = va_arg(args, int);
9                 printf("%d", i);
10            } else if (*format == 'f') {
11                double f = va_arg(args, double);
12                printf("%f", f);
13            }
14        } else {
15            putchar(*format);
16        }
17        format++;
18    }
19    va_end(args);
20 }

```

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 parâmetros:
 - » Um parâmetro pode ser entendido como o que a função precisa para cumprir seu objetivo;
 - » Um parâmetro é uma variável;
 - » 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

```
1 #include <stdio.h>
2 void somar(int a, int b);
3 int main()
4 {
5     return 0;
6 }
7 void somar(int a, int b){
8     int resultado = a + b;
9     printf("%d", resultado);
10 }
```

Código 5: Exemplo função somar declarada e depois definida.

```
1 #include <stdio.h>
2 void somar(int a, int b){
3     int resultado = a + b;
4     printf("%d", resultado);
5 }
6 void main()
7 {
8     return 0;
9 }
```

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 resultado 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");
```

```
1 #include <stdio.h>
2 int somar(int a, int b){
3     int resultado = a + b;
4     return resultado;
5 }
6 int main()
7 {
8     int x,y, soma;
9     printf("Digite um numero:\n");
10    scanf("%d", &x);
11    printf("Digite outro numero:\n");
12    scanf("%d", &y);
13    soma = somar(x,y);
14    printf("Resultado: %d", soma);
15    return 0;
16 }
```

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.

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

Código 9: Exemplo função somar.

Nota:

Uma função pode ser chamada por outra função!

Hierarquia

```
1 #include <stdio.h>
2 #include <math.h>
3 double calcular(double numero){
4     double potencia = pow(numero, 2);
5     double raiz = sqrt(potencia);
6     return raiz;
7 }
8 int main()
9 {
10     double numero,resultado;
11     scanf("%lf", &numero);
12     resultado = calcular(numero);
13     printf("%.2f", resultado);
14     return 0;
15 }
```

Código 10: Função main().

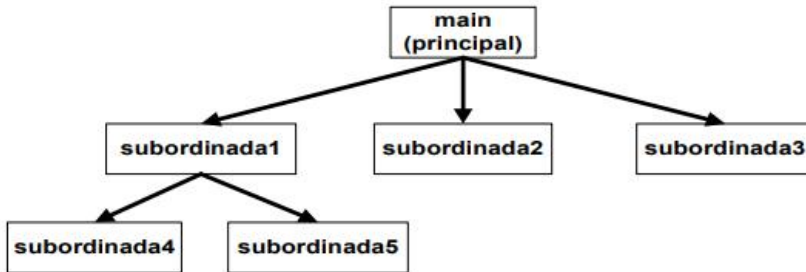


Figura 1: Relacionamento hierárquico de funções (Deitel; Deitel, 2011).

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.

```
1 #include <stdio.h>
2
3 char frase[50] = "esta uma variavel global";
4
5 void mostrar_mensagem(){
6     printf("Dentro da Funcao: %s\n", frase);
7 }
8 int main()
9 {
10     printf("Fora da Funcao: %s\n", frase);
11     mostrar_mensagem();
12     return 0;
13 }
```

Código 11: Exemplo função somar.

Otimização

```
1 #include <stdio.h>
2 int verificar_par(int numero){
3     int eh_par;
4     if(numero % 2 == 0)
5     {
6         eh_par = 0; // VERDADEIRO
7     }else{
8         eh_par = 1; // FALSO
9     }
10    return eh_par;
11 }
12 int main()
13 {
14     int n, validacao;
15     printf("Digite um numero:\n");
16     scanf("%d", &n);
17     validacao = verificar_par(n);
18     if (validacao)
19     {
20         printf("Eh par!\n");
21     }else{
22         printf("Eh impar!\n");
23     }
24     return 0;
25 }
```

Código 12: Exemplo função par.

```
1 #include <stdio.h>
2 int verificar_par(int numero){
3     return numero % 2 == 0;
4 }
5 int main()
6 {
7     int n, validacao;
8     printf("Digite um numero:\n");
9     scanf("%d", &n);
10    validacao = verificar_par(n);
11    if (validacao)
12    {
13        printf("Eh par!\n");
14    }else{
15        printf("Eh impar!\n");
16    }
17    return 0;
18 }
```

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.

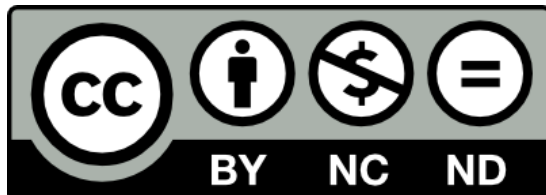


DEITEL, Paul; DEITEL, Harvey. **C: Como Programar**. 6. ed. São Paulo: Pearson Universidades, 2011.



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.

Fundamentos de Programação

Linguagem de Programação C
Função