

## Índice

### 1 – Modularização

- 1.1 Conceituando Funções
- 1.2 Passagem de Parâmetros por Valor e por Referência
- 1.3 Argumentos da linha de comando
- 1.4 Recursividade

Exercícios

### 2 – Ponteiros e Alocação Dinâmica de Memória

- 2.1 Introdução
- 2.2 Manipulação de Ponteiros
- 2.3 Alocação Dinâmica de Memória
- 2.4 Eficiência de Ponteiros em C

Exercícios

### 3 – Estruturas

- 3.1 Conceito
- 3.2 Carga Inicial da Estrutura
- 3.3 Acessando os membros da Estrutura
- 3.4 Vetores de Estruturas
- 3.5 Definição de tipos (typedef)
- 3.6 Passagem de estruturas para funções

Exercícios

# 1. Funções (Modularização)

## 1.1. Conceituando Funções

Elas são importantes em programação estruturada, pois permitem que um código já estabelecido seja reaproveitado quando da nova chamada à função, e que ações específicas possam ser realizadas de forma eficiente. Funções podem, ou não, retornar valores ao ponto do programa que as chamou.

São trechos (ou blocos) de código que diminuem a complexidade de um programa, ou evitam a repetitiva excessiva de determinada parte do aplicativo. Isso se chama **Modularização**. E se aplica às linguagens estruturadas, em que as funções são elementos fundamentais para construção de blocos ordenados e organizados.

Em linguagem C, `main()`, `printf()`, `scanf()`, são funções que apresentam,

A forma geral de função é:

```
<tipo de retorno> nome (parâmetros)
{
    Corpo da função
    Retorno (não obrigatório)
}
```

### EXEMPLO:

A função a seguir retorna a multiplicação de dois valores inteiros recebidos como argumentos:

```
int multiplica(int a, int b) // função de fato
{
    int resultado;
    resultado = a*b;
    return resultado;
}
```

### Definições

- **Parâmetros** são mostrados quando se cria a função (na declaração das variáveis que fazem parte dessa função)
- **Argumentos** tratam do uso efetivo dos elementos passados pela função, quando o programa é executado.
- O retorno da função é o valor que ela desenvolve ao ponto do programa que a chamou; se não for declarado, é do tipo inteiro; void não retorna qualquer valor, ou então não passa parâmetros;
- Funções retornam valores, procedimentos, não;
- Se a função for definida depois de `main()`, é preciso introduzir um protótipo dela antes da função principal.

```

#include <stdio.h>
#include <stdlib.h>
int soma(int a, int b); // prototipo da funcao
int main() {
    int x, y;
    printf("Digite o primeiro numero: \n");
    scanf("%d", &x);
    printf("Digite o segundo numero: \n");
    scanf("%d", &y);
    printf("A soma eh: %d \n", soma(x, y));
    system("pause");
}

int soma(int a, int b) {
    int resultado;
    resultado = a + b;
    return resultado;
}

```

```

C:\Users\Renato\Desktop\teste_livro.exe
Digite o primeiro numero:
3
Digite o segundo numero:
2
A soma eh: 5
Pressione qualquer tecla para continuar. . .
Process returned 0 (0x0) execution time : 6.265 s
Press any key to continue.

```

```

#include <stdio.h>
#include <stdlib.h>
int dif(int a, int b)
{
    int resultado;
    resultado = a - b;
    return(resultado);
}
main()
{
    int x, y, total;
    printf("Quais sao os numeros? \n");
    scanf("%d %d", &x, &y);
    total = dif(x,y);
    printf("Resultado = %d \n", total);
    system("pause");
}

```

```

C:\Users\Renato\Desktop\teste_livro.exe
Quais sao os numeros?
10 5
Resultado = 5
Pressione qualquer tecla para continuar. . .
Process returned 0 (0x0) execution time : 4.856 s
Press any key to continue.

```

```
#include <stdio.h>
#include <stdlib.h>
int max(int n1, int n2) {
    if(n1>n2) return n1;
    else return n2;
}

main() {
    int x, y;
    printf("Digite o primeiro numero: \n");
    scanf("%d", &x);
    printf("Digite o segundo numero: \n");
    scanf("%d", &y);
    printf("O maior eh: %d \n", max(x, y));
    system("pause");
}
```

## 1.2. Argumentos da linha de comando

A função `main()` tem, em sua definição completa a possibilidade de passar argumentos para a linha de comando do sistema operacional.

Por ser a primeira função a ser chamada pelo programa, seus argumentos são passados junto com o comando que executa o programa.

Ainda temos os argumentos **`argc`** e **`argv`** mostram respectivamente, **o número de argumentos passado** para o programa e os valores de tais argumentos.

O parâmetro **`argv`** é, na verdade, **um vetor de strings** que armazena o nome do programa e os argumentos em forma de cadeia de caracteres;

**`argv[0]` é o nome do programa,**

**`argv[1]`, o primeiro argumento passado ao programa, e assim por diante.**

A função `main()` retorna um número inteiro para o processo que a chamou – geralmente, o próprio sistema operacional. Podemos declarar `main()` como `void`, se não retornar nenhum valor.

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

int main(int argc, char *argv[]) // aqui a definição completa de main()
{
    printf("Numero de argumentos de main(): %d \n", argc);
    printf("Valores dos argumentos de main(): %s e %s \n", argv[0], argv[1]);
    return 0;
}
```

### 1.3. Passagem de Parâmetros

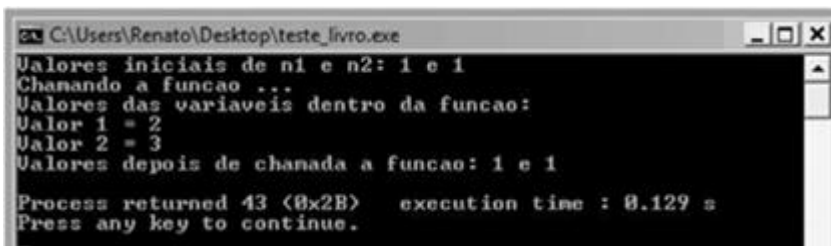
A **Chamada por Valor** é a passagem normal do valor dos argumentos para a função. Os valores dos argumentos passados não são modificados, pois é fornecida uma cópia dos valores para a função.

Na **Chamada por Referência** são passados os endereços de memória dos argumentos. Portanto, os valores podem ser modificados.

```
/* Função com chamada por valor */

#include <stdio.h>
#include <stdlib.h>
int valor(int a, int b)
{
    a = a + 1; /* primeiro argumento foi modificado */
    b = b + 2; /* segundo argumento foi modificado */
    printf("Valores das variaveis dentro da funcao: \n");
    printf("Valor 1 = %d \n", a);
    printf("Valor 2 = %d \n", b);
}

main()
{
    int n1 = 1, n2 = 1, total;
    printf("Valores iniciais de n1 e n2: %d e %d \n", n1, n2);
    printf("Chamando a funcao ... \n");
    valor(n1, n2);
    printf("Valores depois de chamada a funcao: %d e %d \n", n1, n2);
}
```

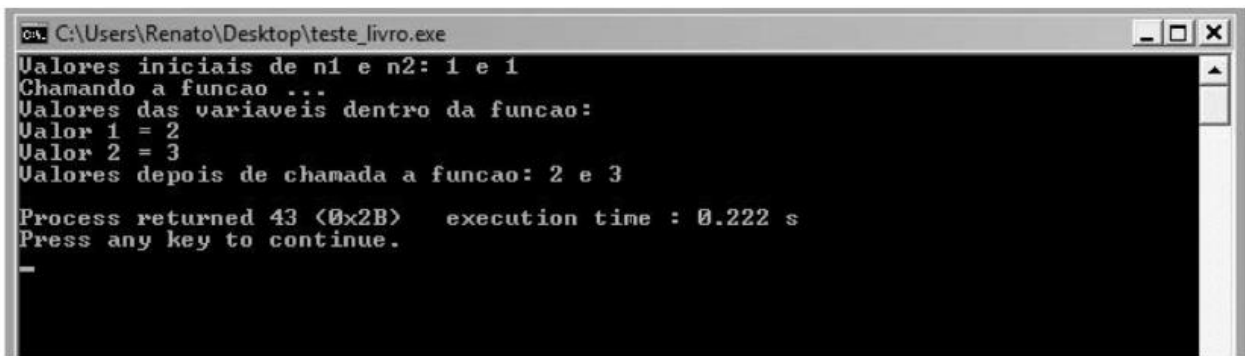


```
C:\Users\Renato\Desktop\teste_livro.exe
Valores iniciais de n1 e n2: 1 e 1
Chamando a funcao ...
Valores das variaveis dentro da funcao:
Valor 1 = 2
Valor 2 = 3
Valores depois de chamada a funcao: 1 e 1
Process returned 43 (0x2B)   execution time : 0.129 s
Press any key to continue.
```

**NOTE QUE NÃO HOUE  
A ALTERAÇÃO DOS VALORES  
DAS VARIÁVEIS N1 E N2 !!**

```
#include <stdio.h>
#include <stdlib.h>
int valor(int *a, int *b)
{
    *a = *a + 1; /* primeiro argumento foi modificado */
    *b = *b + 2; /* segundo argumento foi modificado */
    printf("Valores das variaveis dentro da funcao: \n");
    printf("Valor 1 = %d \n", *a);
    printf("Valor 2 = %d \n", *b);
}

main()
{
    int n1 = 1, n2 = 1, total;
    printf("Valores iniciais de n1 e n2: %d e %d \n", n1, n2);
    printf("Chamando a funcao ... \n");
    valor(&n1, &n2); // passado o endereco da variavel
    printf("Valores depois de chamada a funcao: %d e %d \n", n1, n2);
}
```



```
C:\Users\Renato\Desktop\teste_livro.exe
Valores iniciais de n1 e n2: 1 e 1
Chamando a funcao ...
Valores das variaveis dentro da funcao:
Valor 1 = 2
Valor 2 = 3
Valores depois de chamada a funcao: 2 e 3

Process returned 43 (0x2B)   execution time : 0.222 s
Press any key to continue.
-
```

No próximo exemplo, apresentamos o clássico algoritmo de troca de valores entre variáveis (swap); para que valores enviados para tal função possam retornar ao ponto de chamada – já trocados – é preciso passá-los por referência, e não por valor.

```
// Função de troca de valores
void swap (int * x, int * y)
{
    int temp;
    temp = *x;
    *x = *y;
    *y = temp;
}
```

Na função main( ) utilizamos a chamada swap (&i, &j), em que “&” representa o endereço da variável, e não seu valor.

Importante:

- As strings e vetores são sempre chamadas por referência. Quando a Linguagem C passa um vetor ou uma string para uma função, o que se passa, na verdade, é o endereço de memória inicial do mesmo;
- As variáveis declaradas dentro de uma função são locais, visíveis apenas ali e destruídas após o término da função. Caso se deseje manter o valor da variável entre as chamadas a uma função, ela deve ser declarada static.

## 1.4. Recursividade

É a possibilidade de uma função chamar a si mesma.

Knuth, por exemplo, apresenta o algoritmo (de Euclides) de determinação do MDC (Máximo Divisor Comum – o maior inteiro positivo que divide dois números inteiros positivos,  $m$  e  $n$ ) como recursivo:

[Achar o resto da divisão] Dividir  $m$  por  $n$  e armazenar o resto da divisão em  $r$  (temos que  $0 \leq r \leq n$ ).

[É zero?] Se  $r = 0$ , o algoritmo termina;  $n$  é a resposta.

[Reduzir] Faça  $m = n$ ,  $n = r$ , e volte à etapa 1.

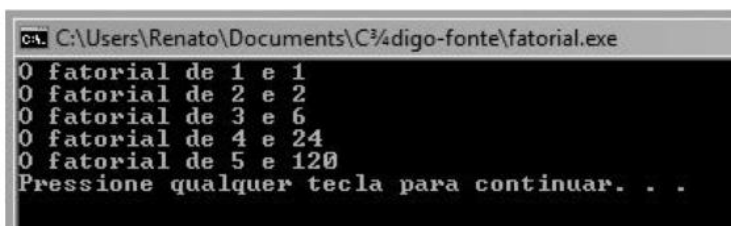
A implementação em Linguagem C poderia ser assim:

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

int mdc(int m, int n) {
    int r = m%n;
    if(r == 0)
        return n;
    return mdc(n, (m%n));
}

main() {
    int a=544; int b=119;
    printf("%d \n", mdc(a, b));
    system("pause");
}
```

```
#include <stdio.h>
#include <stdlib.h>
int fatorial(int valor) {
    if(valor == 1); return(1);
    else return(valor * fatorial(valor -1));
}
main() {
    int i;
    for(i=1; i <= 5; i++)
        printf("O fatorial de %d e %d \n", i, fatorial(i));
}
```



```
C:\Users\Renato\Documents\C3\4digo-fonte\fatorial.exe
O fatorial de 1 e 1
O fatorial de 2 e 2
O fatorial de 3 e 6
O fatorial de 4 e 24
O fatorial de 5 e 120
Pressione qualquer tecla para continuar. . .
```

O que explica a recursividade dessa função é o fato de que o fatorial de 5 é igual a  $5 \times 4 \times 3 \times 2 \times 1$ . Ou seja:

- fatorial de 5 =  $5 * \text{fatorial de } 4$
- fatorial de 4 =  $4 * \text{fatorial de } 3$
- fatorial de 3 =  $3 * \text{fatorial de } 2$
- fatorial de 2 =  $2 * \text{fatorial de } 1$

Ou seja, o fatorial de um número = número \* (fatorial do número - 1).

## Exercícios

1. Escreva um programa que implemente uma função que retorne a diferença entre dois números inteiros digitados pelo usuário.
2. Escreva uma função que retorne a divisão entre dois números. Atenção para a questão da divisão por zero! A divisão não pode ocorrer se o divisor for zero.
3. Escreva um programa que calcule a área de um círculo a partir de uma função especialmente desenhada para isso; essa função recebe o valor do raio e retorna a área do círculo.
4. Crie um sistema de caixa eletrônico, utilizando menus (switch) e outros recursos, que realizem operações financeiras a partir de funções especificamente projetadas para tal. Lembre-se de que o caixa eletrônico é um programa que roda como repetição contínua, até que o usuário decida encerrar as operações. Operações: depósito, Saque e Saldo.
5. Crie uma função que determine se dado caractere está entre 'a' e 'z'. Dica: Use a tabela ASCII.
6. Escreva um programa que implemente uma função que passe dado número inteiro como parâmetro, e este desenhe um número equivalente a "\*" na tela.
7. Escreva uma função que retorne o cubo do valor passado como argumento.