Algoritmos e Estruturas de Dados I

Modularização – Função e Procedimento

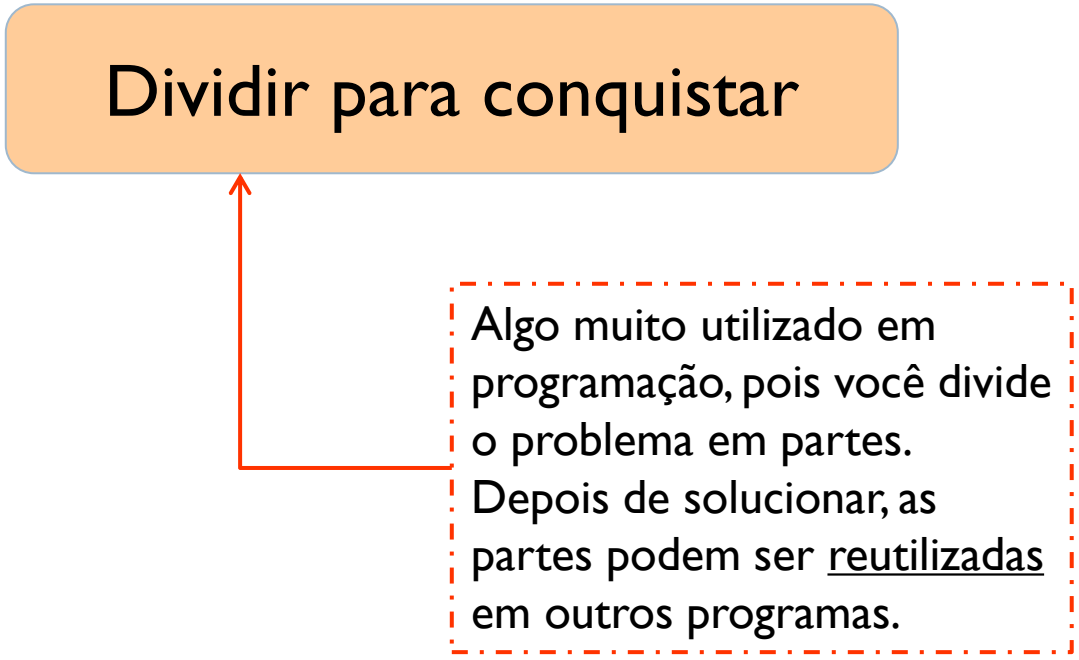
Prof. Ivre Marjorie

Introdução

- Função: é um conjunto de instruções desenhadas para cumprir uma tarefa particular e agrupadas numa unidade com um nome para referenciá-las
- Exemplos:
  - Função para imprimir um valor na tela
  - Função para preencher um vetor
  - Função para fazer uma soma de vários valores
  - Função para calcular a raiz de um número
- Você pode criar uma função para fazer qualquer coisa no seu programa

Introdução

- **Razão principal** para usar funções: dividir a tarefa original em pequenas tarefas que simplificam e organizam o programa como um todo



Introdução

- **Outra razão**: reduzir o tamanho do programa
- Qualquer sequência de instruções que apareça no programa mais de uma vez é candidata a ser uma função
- O código de uma função é agregado ao programa um única vez e pode ser executado muitas vezes no decorrer do programa
  - Para isso, basta fazer uma chamada da função

Função x Procedimento

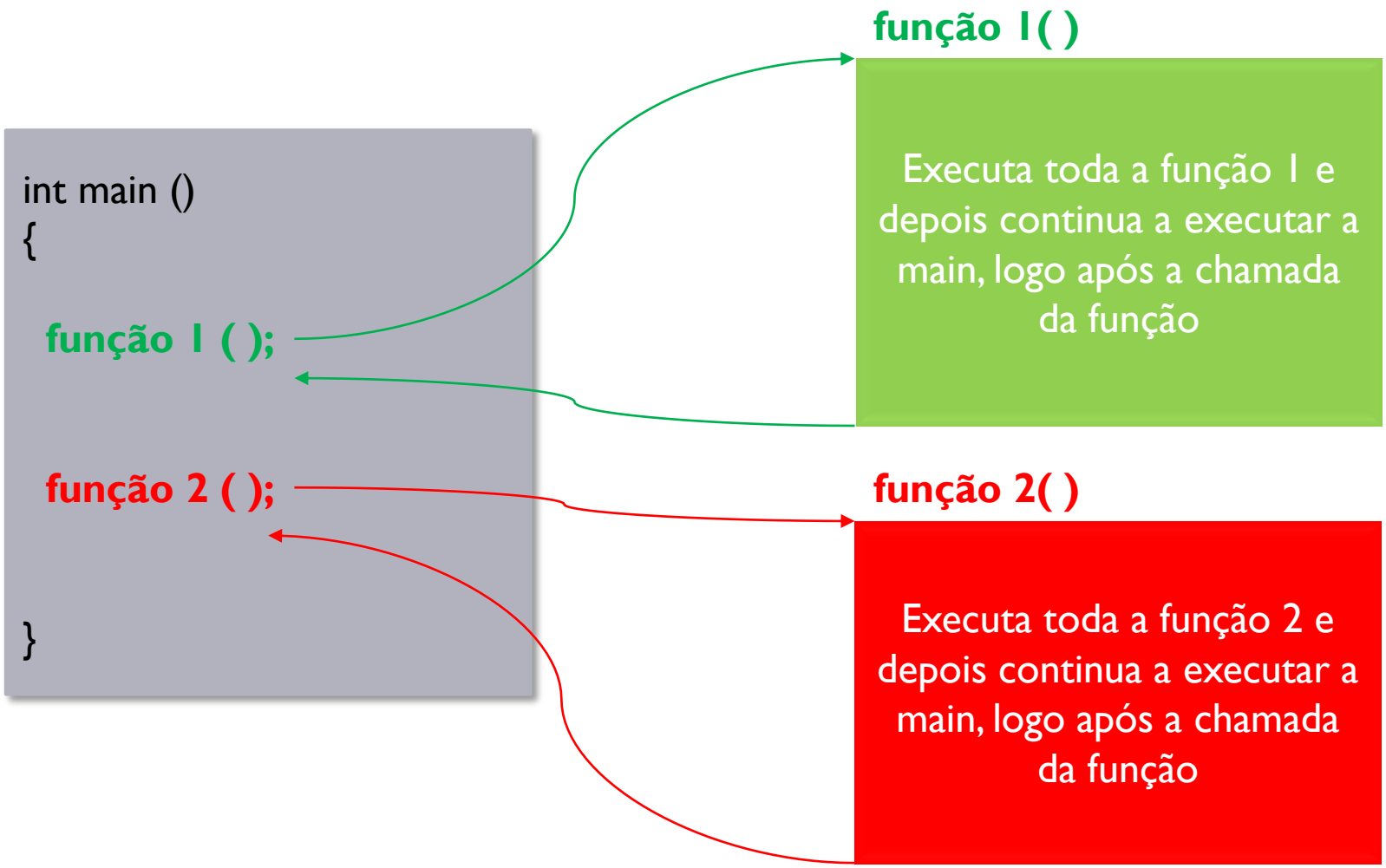
- **Função:** retorna algum resultado
  - Será de algum tipo
    - int, float, double, string, char, etc
  - deverá conter no bloco de comando o comando **return**

- **Procedimento (função sem retorno):** não retorna nenhum valor
  - Será do tipo void e não terá nenhum retorno

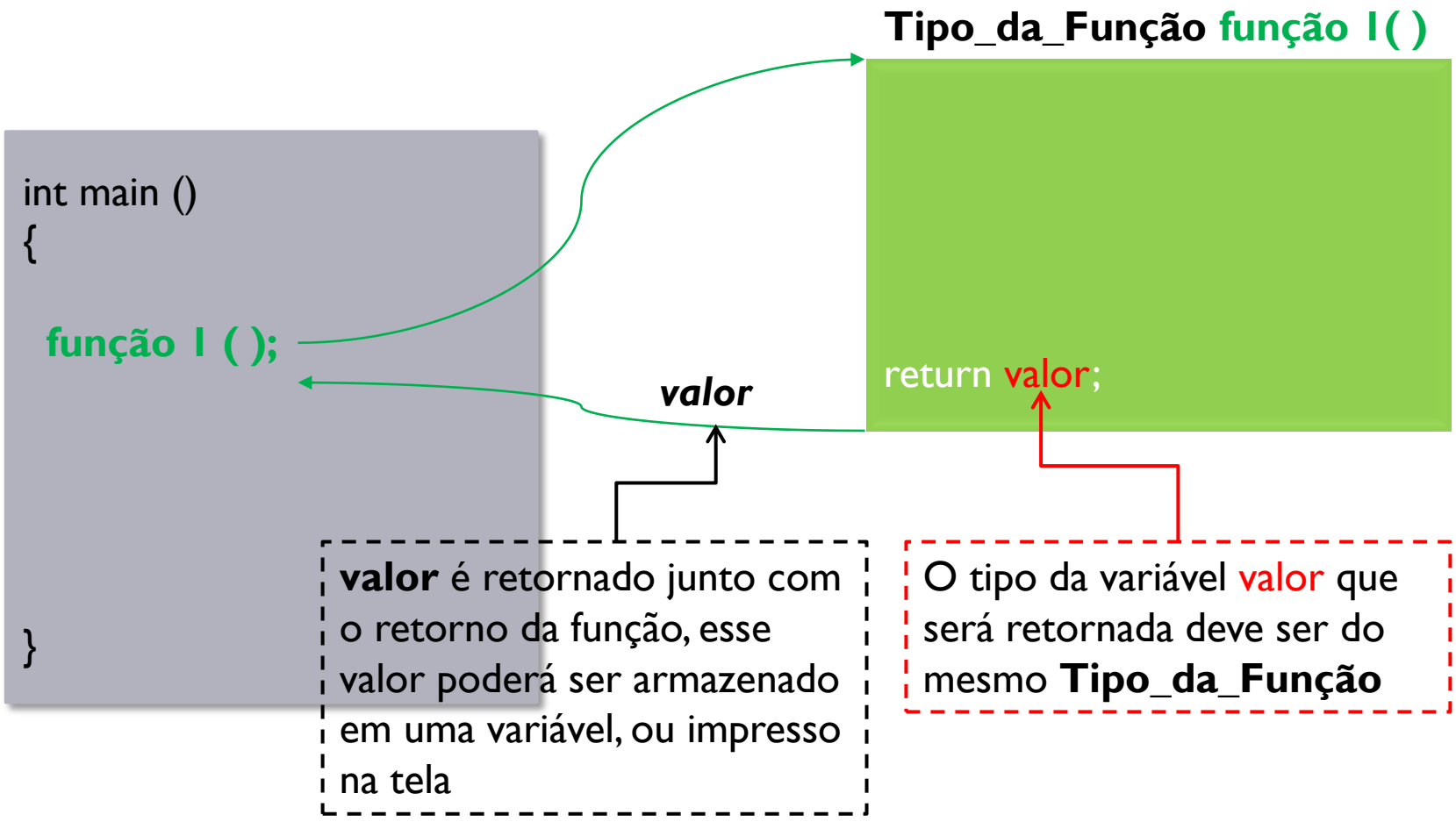
Chamando uma função

- Um programa pode conter uma ou mais funções, das quais uma delas deve ser a função **Main (Função Principal)**
- A execução do programa sempre começa na função **Main** e,
  - quando o controle do programa encontra uma instrução que inclui o nome de uma função, a função é chamada
  - Depois que essa função é executada o programa continua a executar a main a partir de onde parou
  - A figura a seguir busca representar a chamada de uma função
  - É possível termos chamadas de função dentro de outras funções, e o processo de execução será o mesmo

Chamando uma função



Chamando uma função: Função com retorno



Protótipo de Função

- Uma função não pode ser chamada sem antes ter sido declarada
- A declaração de uma função é dita ***protótipo da função***
  - Deve ser colocada no início do programa, antes da main(), e estabelece o tipo da função e os argumentos que ela recebe
- O protótipo é exatamente a primeira linha da criação da função, com ponto e vírgula no final

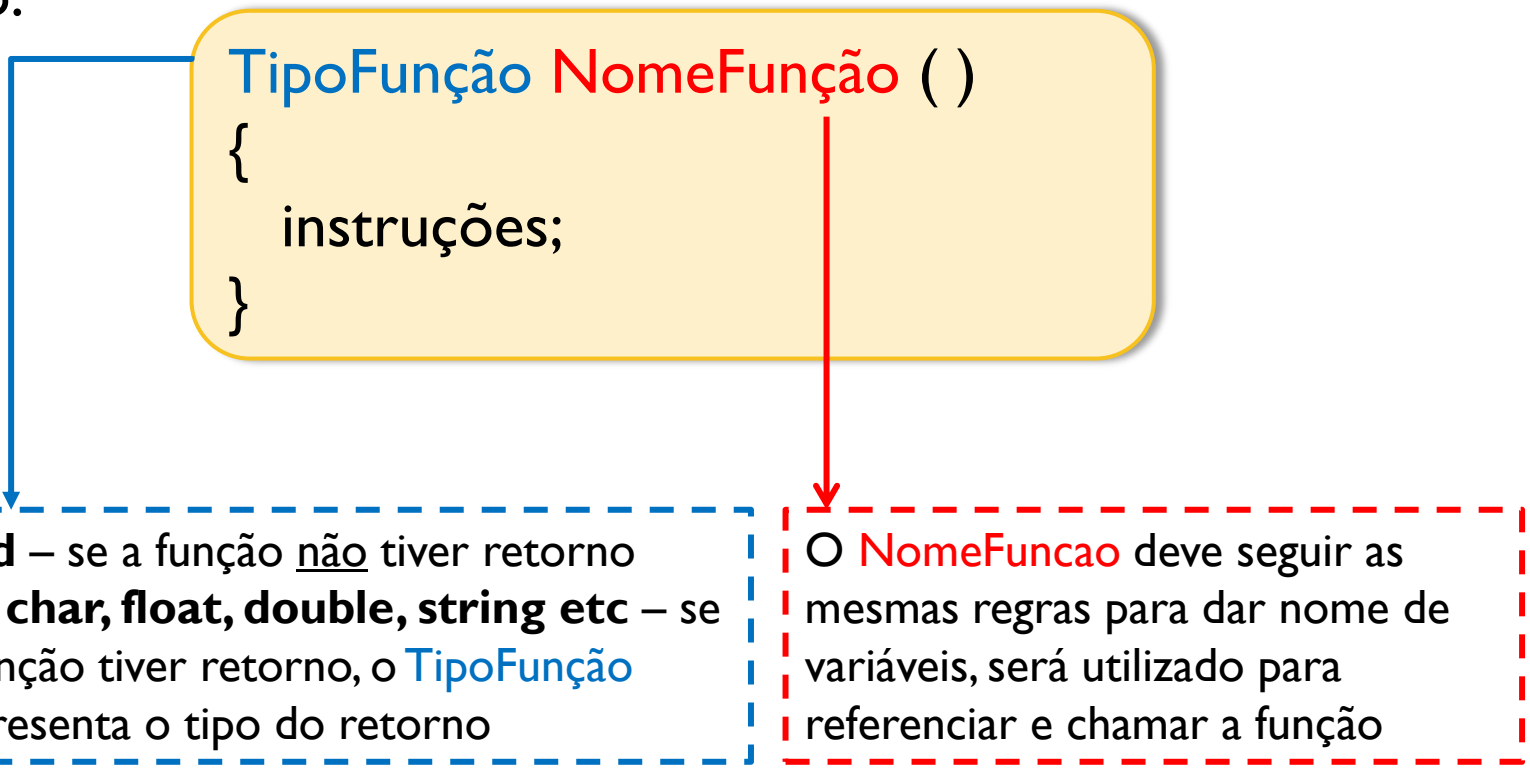
```
TipoFunção NomeFunção ( );
```

- Ou

```
TipoFunção NomeFunção ( argumentos );
```

Criação da Função

- As funções devem ser criadas antes ou depois da função principal, a main( )
- Além disso, as funções devem ter um nome e um tipo, como abaixo:



Tipos de Funções

- É possível construir as seguintes funções:
  1. Função com retorno e sem parâmetro
  2. Função com retorno e com parâmetro
  3. Função (procedimento) sem retorno e sem parâmetro
  4. Função (procedimento) sem retorno e com parâmetro

1- Função com retorno e sem parâmetro

- Essa função não recebe nenhum argumento (valor), mas retorna um valor (que geralmente, foi calculado dentro da função)
- Exemplo:

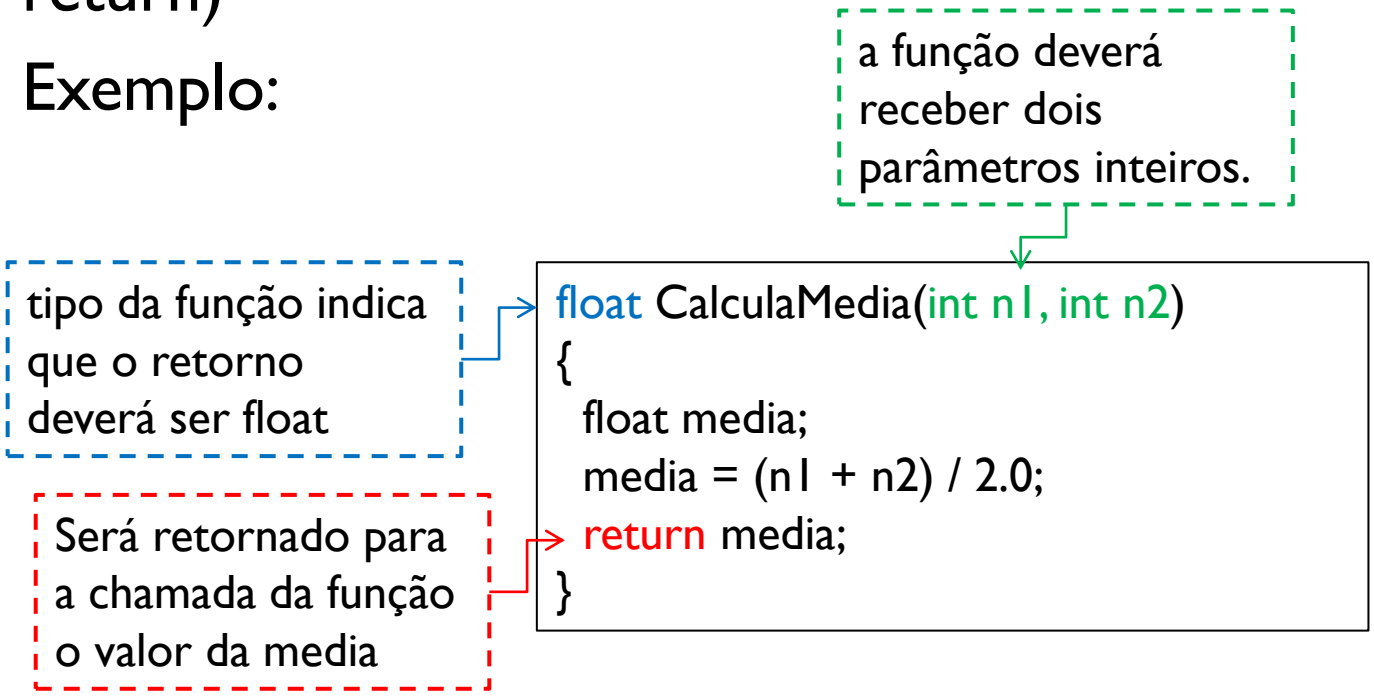
tipo da função indica que o retorno deverá ser float

```
float CalculaMedia()
{
  float num1, num2, media;
  printf("Digite o primeiro numero:");
  scanf("%f", &num1);
  printf("Digite o segundo numero:");
  scanf("%f", &num2);
  media = (num1 + num2)/2;
  return media;
}
```

Será retornado para a chamada da função o valor da media

2- Função com retorno e com parâmetro

- Essa função recebe um ou mais parâmetros/argumentos e retorna um valor (é necessário o uso da palavra chave return)
- Exemplo:



3- Função sem retorno e sem parâmetro

- Essa função não recebe nenhum argumento (valor), e não retorna um valor (será sempre do tipo **void**)
- Exemplo:

O tipo void indica que a função não irá retornar um valor

```
void CalculaMedia()
{
  float num1, num2, media;
  printf("Digite o primeiro numero:");
  scanf("%f", &num1);
  printf("Digite o segundo numero:");
  scanf("%f", &num2);
  media = (num1 + num2)/2;
  printf("A media e: %.2f", media);
}
```

4- Função sem retorno e com parâmetro

- Essa função recebe um ou mais parâmetros/argumentos, mas não retorna um valor (será do tipo void)
- Exemplo:

a função deverá receber dois parâmetros inteiros.

O tipo void indica que a função não irá retornar um valor, portanto, o valor calculado deverá ser impresso aqui mesmo

```
void CalculaMedia(int n1, int n2)
{
  float media;
  media = (n1 + n2) / 2.0;
   printf("A media e: %.2f", media);
}
```

Passagem de parâmetros

- Uma função pode receber um ou mais valores, chamados de parâmetros (ou argumentos)
- No entanto, na criação da função é necessário indicar os tipos de dados desse(s) parâmetro(s)
- Na chamada da função, não é necessário indicar o tipo de dado, apenas passar o valor e/ou a variável
- A passagem de parâmetro pode ser:
  - Por valor
  - Por referência

Parâmetros por Valor

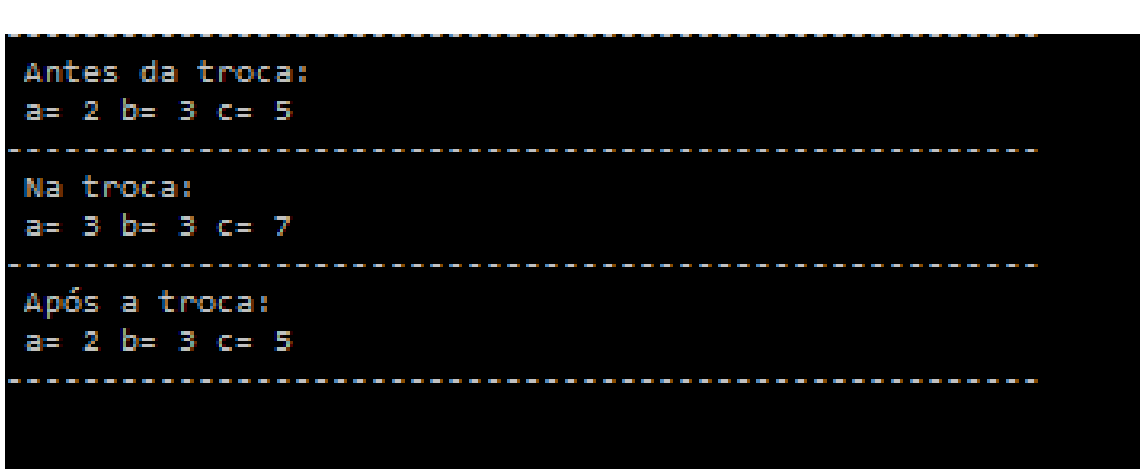
- Passar uma variável como parâmetro para uma função por valor significa passar uma cópia da variável para a função
- Quaisquer alterações que ocorrem dentro da função para o parâmetro não tem nenhum efeito nos dados originais armazenados na variável
- Portanto, as alterações dentro da função não são refletidas no ambiente da sua chamada

Exemplo

```
void troca(int a, int b, int c);
int Main()
{
  int a = 2, b = 3, c = 5;
  printf("\n");
  printf("-----------------------------------------------------");
  printf("\n Antes da troca: \n a= %d b= %d c= %d ", a, b, c);
  printf("\n");
  printf("-----------------------------------------------------");
  troca(a, b, c);
  printf("\n Apos a troca: \n a= %d b= %d c= %d ", a, b, c);
  printf("\n");
  printf("-----------------------------------------------------");
  return 0;
}
void troca(int a, int b, int c)
{
  a = 3;
  c = a + 4;
  printf("\n Na troca: \n a= %d b= %d c= %d ", a, b, c);
  printf("\n");
  printf("-----------------------------------------------------");
}
```

Exemplo

- Resultado:



Observe que o valor de todas as variáveis a, b e c são os mesmos antes e depois da função troca

Parâmetros por Referência

- Passagem por referência permite que membros da função alterarem o valor dos parâmetros e
  - essa alteração persista no ambiente de chamada
- Para passar um parâmetro por referência na linguagem C, use os seguintes operadores:
  - Na chamada da função use o operador &
  - Na criação da função use o operador *
- No protótipo, basta indicar o tipo de dado e o operador * para as variáveis que serão referência

Parâmetros por Referência

- Exemplo:

```
void CalculaMedia(int n1, int n2, float * );
int main()
{
  int num1, num2, media;
  printf("Digite o primeiro valor:")
  scanf("%d", &num1);
  printf("Digite o segundo valor:")
  scanf("%d", &num2);
  CalculaMedia(num1, num2,  &media);
  printf("A media e:  ", media);
  return 0;
}
void CalculaMedia(int n1, int n2, float *m)
{
   m = (n1+ n2 ) / 2;
}
```

Nesse exemplo, a média não é retornada na função CalculaMedia(), mas como o parâmetro é por referência, qualquer alteração no valor da média dentro da função é refletido no ambiente da chamada. Observe que no protótipo apenas indicamos com tipo + * a variável que será parâmetro por referência

Na chamada da função usamos o operador &, observe que apenas a variável média foi passada por referência

Na criação da função usamos o operador *

Exemplo

```
double entradanumeros();
void calculamedia(double num1, double num2, double *);
```

```
int main()
{
   double numa, numb, med = 0;
   numa = entradanumeros();
   numb = entradanumeros();
   calculamedia(numa, numb, & med);
   printf("\n A media é:&.2lf", med);
   return 0;
}
```

```
double entradanumeros()
{
   double x;
   printf("Digite um numero: ");
   scanf("%lf", x);
   return x;
}
```

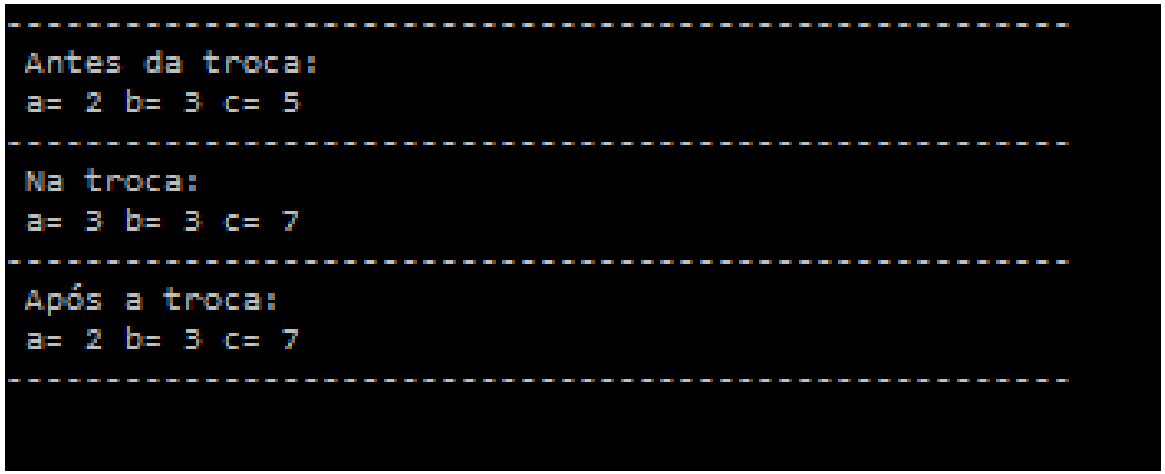
```
void calculamedia(double num1, double num2, double *media)
{
   media = (num1 + num2) / 2;
}
```

Exemplo

```
void troca(int a, int b, int *);
int Main()
{
  int a = 2, b = 3, c = 5;
  printf("-----------------------------------------------------");
  printf(" Antes da troca: \n a= %d b= %d c= %d ", a, b, c);
  printf("\n");
  printf("-----------------------------------------------------");
  troca(a, b, &c);
  printf(" Apos a troca: \n a= %d b= %d c= %d ", a, b, c);
  printf("\n");
  printf("-----------------------------------------------------");
  return 0;
}
void troca(int a, int b, int *c)
{
  a = 3;
  c = a + 4;
  printf(" Na troca: \n a= %d b= %d c= %d ", a, b, c);
  printf("\n");
  printf("-----------------------------------------------------");
}
```

Exemplo

- Resultado:



Observe que o valor da variável c foi alterado dentro da função troca e isso foi refletido na **função principal**, ou seja, após a troca

Vetor e Matriz como parâmetros

- O vetor e matriz já são considerados referências e **não** precisam do operador **&** na passagem do parâmetro
- Com isso, qualquer alteração em um vetor e matriz dentro de uma função será refletida no ambiente que chamou a função
- Na chamada da função indicamos apenas o nome do vetor e/ou matriz
- Já na criação da função indicamos que o parâmetro é vetor ou matriz com o uso dos colchetes
  - No caso dos vetores não é necessário colocar o tamanho
  - Já no caso das matrizes, é necessário indicar o tamanho da segunda dimensão (ou seja, das colunas)

Exemplo

```
void entradavetor(int v[ ]);
void somavetor(int v[ ], int * );
void imprimevetor(int v[ ]);

int main()
{
   int vet[6];
   int soma = 0;
   entradavetor(vet);
   somavetor(vet, &soma);
   printf("A soma dos valores é: %d",soma);
   imprimevetor(vet);
   return 0;
}
```

Observe:

- ✓ a soma teve que ser referenciada (&) e (*) mas o vetor não.
- ✓ o vetor é criado (declarado) dentro da função principal (main)

```
void entradavetor(int v[ ])
{
   int i;
   for(i=0; i<6; i++)
    {
       printf("Digite um numero: ");
       scanf("%d", &v[ i ]);
    }
}
void somavetor(int v[ ], int *s)
{
   int i;
   for(i = 0; i < 6; i++)
    {
       s = s + v[ i ];
    }
}
void imprimevetor(int v[ ])
{
   int i;
   printf("Os valores do vetor são: ");
   for(i = 0; i < 6; i++)
    {
       printf(" %d | ",v[ i ]);
    }
}
```

Exemplo livro – parte 1

```c
/* histograma.c */
/* Mostra matriz de duas dimensões como argumento de função */
#include <stdio.h>
#include <stdlib.h>
#define MES 3
#define FUNC 5

void histograma(int [][MES], int); /* Protótipo */

int main()
{
   int pecas[FUNC][MES], i, j;

   for(i=0; i<FUNC ; i++)
          for(j=0; j<MES ; j++)
          {
                 printf("Funcionário: %d\tmês %d: ", i+1, j+1);
                 scanf("%d", &pecas[i][j]);
          }
   histograma(pecas,FUNC);
   system("PAUSE");
   return 0;
}
```

Fonte: Livro Treinamento em Linguagem C – páginas 198 e 199

Exemplo livro – parte 2

```
/* Imprime um histograma horizontal */
void histograma(int pecas[][MES], int nfunc)
{
   const float MAXBARRA=50.0;
   int max=0,temp=0, i, j, tam;

   for(i=0; i<nfunc; i++)
   {
         for(j=0; j<MES ; j++)
            temp += pecas[i][j];
         if(max < temp)      max=temp;
   }

   temp=0;
   for(i=0 ; i<nfunc; i++)
   {
         for(j=0; j<MES ; j++) temp += pecas[i][j];
         printf("%2d - %5d:", i+1, temp);

         tam = (int)((float)temp/(float)max*MAXBARRA);

         for(j=0; j<tam; j++) printf("*");
```

Fonte: Livro Treinamento em Linguagem C – páginas 198 e 199

Exemplo livro – parte 3

```
            printf("\n");
    }
}
```

Eis um exemplo da execução do programa:

```
Funcionário 1 mês 1: 1144
Funcionário 1 mês 2: 1200
Funcionário 1 mês 3: 1200
Funcionário 2 mês 1: 800
Funcionário 2 mês 2: 630
Funcionário 2 mês 3: 750
Funcionário 3 mês 1: 2345
Funcionário 3 mês 2: 2400
Funcionário 3 mês 3: 2567
Funcionário 4 mês 1: 1789
Funcionário 4 mês 2: 1876
Funcionário 4 mês 3: 1654
Funcionário 5 mês 1: 3456
Funcionário 5 mês 2: 3214
Funcionário 5 mês 3: 2999

1 -  3544:******
2 -  5724:**********
3 - 13036:************************
4 - 18355:**********************************
5 - 28024:**************************************************
```

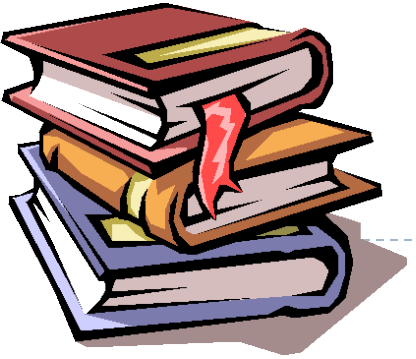
Fonte: Livro Treinamento em Linguagem C – páginas 198 e 199

Exercícios

1- Faça uma função que calcule receba 3 números como parâmetros, ordene os números em ordem crescente e apresente-os na tela.

2- Faça uma função que verifique se um número é divisível por outro sem usar o operador %.

3- Faça um programa que preencha um vetor na função principal. Crie uma função que receba um vetor como parâmetro e ordene crescentemente. Crie uma função para imprimir o vetor.



Referência Bibliográfica

- MIZRAHI, Victorine Viviane. **Treinamento em linguagem C.** São Paulo:Pearson Prentice Hall, 2008. 2ª edição. Curso Completo. Capítulos 5, 7 e 9.

- ASCENCIO, Ana Fernanda Gomes e CAMPOS, Edilene A. Veneruchi. **Fundamentos da Programação de Computadores – Algoritmos, Pascal, C/C++ e Java**. São Paulo: Pearson Prentice Hall, 2012. 3ª Edição.