

# **Algoritmos e Programação II**

## **Subrotinas em C (Funções)**

Profa. Dra. Eloize Seno

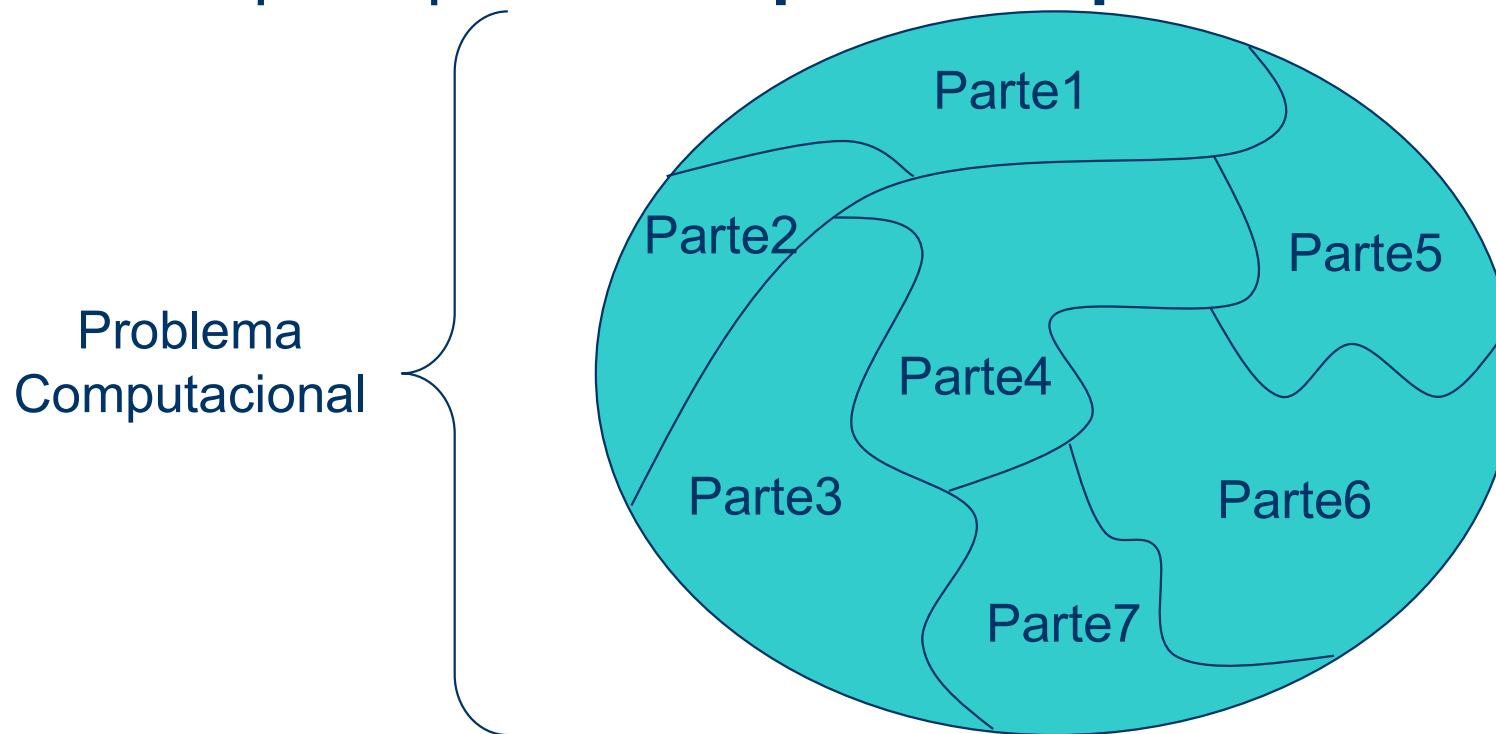


# Sub-rotinas (ou subprogramas)

- São blocos de instruções que realizam tarefas específicas.
- Permitem o **reuso**:
  - O código desse BLOCO é carregado uma vez e executado N vezes pelo programa.
- Permitem a **modularização**:
  - O uso de sub-rotinas tende a diminuir o tamanho dos programas, pois os trechos de programas são identificados e isolados em BLOCO de instruções.

# Sub-rotinas (ou subprogramas)

- Modularização:
  - Colabora com a solução de problemas por meio do princípio **dividir para conquistar**



# Sub-rotinas (ou subprogramas)

- Modularização:
  - Ao se deparar com o problema computacional gigantesco, dificuldades na elaboração da solução irão surgir. No entanto, no desenvolvimento de software, profissionais com mais experiência em determinados tipos de problemas são encontrados nas equipes e, uma fração do problema, será a ele encaminhada para que a solução seja implementada.

# Sub-rotinas (ou subprogramas)

- **Outras vantagens:**
  - Teste
  - Depuração
  - Manutenção

# Sub-rotinas (ou subprogramas)

- Conceitos importantes quando se trabalha com sub-rotinas:
  - Variável local;
  - Variável global;
  - Passagem de parâmetros;
  - Valor de retorno.

# Sub-rotinas em C: Funções

- Todo programa em C tem no mínimo uma função: a função **main()**
- Outras funções pré-definidas da linguagem: `gets()`, `strcpy()`, `clrscr()`, `strcmp()`, `printf()`, etc.
- Podemos definir nossas próprias funções :)

# Funções em C

- Forma Geral:

```
Tipo nome_funcao(lista de parametros)
{
    corpo da função
    return ret; // o valor de retorno é opcional
}
```



# Funções em C

- Forma Geral:

Declaração dos parâmetros:  
Tipo1 var1, tipo2 var2 ... tipoN varN

int  
char  
float  
...

**Tipo** nome\_funcao(lista de parametros)

{

corpo da função

return **ret**;

}

Encerra a execução  
da função

# Funções em C

- Podem ser definidas **antes** ou **depois** da função **main**
- Antes da função **main**: **nenhum cuidado especial**
- Depois da função **main**: **usar protótipos de função**
  - o protótipo (ou assinatura) é o cabeçalho da função e deve ser inserido antes do **main** e finalizado com (;)

# Funções sem passagem de parâmetros e sem retorno

```
#include <stdio.h>
void soma()
{
    int a, b, s;
    printf("\nDigite dois numeros: ");
    scanf("%d%d", &a, &b);
    s = a + b;
    printf("\nSoma: %d", s);
}

int main ( )
{
    soma(); // chama a função soma
}
```

# Funções sem passagem de parâmetros e sem retorno

```
#include <stdio.h>
void soma()
{
    int a, b, s;
    printf("\nDigite dois numeros: ");
    scanf("%d%d", &a, &b);
    s = a + b;
    printf("\nSoma: %d", s);
}
int main ( )
{
    soma(); // chama a função soma
}
```

Variáveis  
locais

Não há  
parâmetros!

# Funções com passagem de parâmetros e sem retorno

```
#include <stdio.h>
void media(int num1, int num2)
{
    float media;
    media = (num1 + num2) / 2;
    printf("\nMedia: %2.2f", media);
}
int main ( )
{
    int n1, n2;
    printf("\nDigite dois numeros: ");
    scanf("%d%d", &n1, &n2);
    media(n1, n2); /* chama a função media,
                    passando n1 e n2*/
}
```

Parâmetros da função

# Funções sem passagem de parâmetros e com retorno

```
#include <stdio.h>
float produto ()
{
    float n1, n2;
    printf("\nDigite dois numeros: ");
    scanf("%d%d", &n1, &n2);
    return (n1 * n2); // retorno da função
}

int main ( )
{
    float prod;
    prod = produto(); /* chama a função produto e
                       guarda o resultado em prod*/
    printf("\nProduto = %2.2f", prod);
}
```

Variáveis  
locais

# Funções com passagem de parâmetros e com retorno

```
#include <stdio.h>
float divisao (int dividendo, int divisor)
{
    return (dividendo/divisor); // retorno da função
}

int main ( )
{
    int n1, n2;
    float res;
    printf("\nDigite dois numeros: ");
    scanf("%d%d", &n1, &n2);
    res = divisao(n1, n2); /*chama a função divisao
                           e guarda o resultado em res*/
    printf("\nResultado da divisao = %2.2f", res);
}
```

# Criação da função após a função main() - Exemplo

```
#include <stdio.h>
// assinatura (ou prototipo) da função
float divisao (int dividendo, int divisor);
int main ( )
{
    // programa principal
}
float divisao (int dividendo, int divisor)
{
    float r;
    r = (dividendo/divisor);
    return r;
}
```



# Exercício de fixação

- Crie um programa para resolver equações do 2º grau. Implemente uma função para calcular o delta, uma função para calcular uma raiz real e outra função para calcular duas raízes reais. Considere:

$ax^2 + bx + c = 0$  **Obs:** a deve ser diferente de 0

$\text{delta} = b^2 - 4 * a * c$

Se  $\text{delta} < 0$ , não existe raiz real

Se  $\text{delta} = 0$ , existe uma raiz real:

$x = (-b) / (2 * a)$

Se  $\text{delta} > 0$ , existem duas raízes reais:

$x1 = (-b + \text{raiz quadrada de delta}) / (2 * a)$

$x2 = (-b - \text{raiz quadrada de delta}) / (2 * a)$

**Atenção:** Os resultados produzidos por cada função deverão ser impressos pela função **main()** e nunca dentro das funções. **Não use variáveis globais.**

# Casos de Testes do Programa

- Testes

1:  $\langle 1, 1, 1; \text{não tem raízes} \rangle$

2:  $\langle 1, 2, 1; -1 \rangle$

3:  $\langle 1, 0, -4; 2, -2 \rangle$

# Passagem de Parâmetros

```
#include <stdio.h>
#include <stdlib.h>
void soma (int i, int j, int *k){
    *k = i + j;
}
int main() {
    int n1, n2, s = 0;
    printf("N1 = ");
    scanf("%d", &n1);
    printf("N2 = ");
    scanf("%d", &n2);
    soma(n1, n2, &s);
    printf("Soma = %d\n", s);
    system("PAUSE");
}
```

A função soma pode ser interpretada como sendo uma caixa preta que possui três parâmetros, sendo os dois primeiros de entrada e o terceiro de saída.

# Passagem de Parâmetros

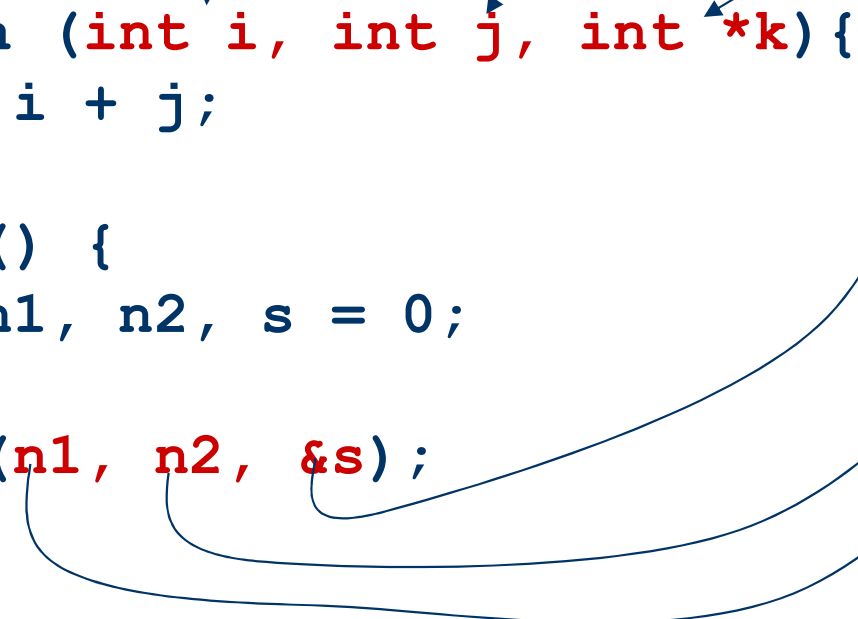
- Neste exemplo é possível visualizar a função soma e a função principal;
- A função soma possui a seguinte assinatura:  
`void soma (int i, int j, int *k)`
- As variáveis i, j e k são declaradas como parâmetros, que podem ser classificadas em:
  - Passagem por valor (i e j);
  - Passagem por referência (k);
- Qual o significado disso?



# Passagem de Parâmetros

- Quando a chamada da função é executada, os parâmetros (VALORES) são transferidos do programa principal para a função;

```
void soma (int i, int j, int *k) {  
    *k = i + j;  
}  
  
int main() {  
    int n1, n2, s = 0;  
    //...  
    soma (n1, n2, &s);  
    //...  
}
```



Os valores são transferidos do programa principal para a função soma

# Passagem de Parâmetros

- Observe a assinatura e passagem de parâmetros:

```
void soma (int i, int j, int *k)
```

```
soma (n1, n2, &s) ;
```

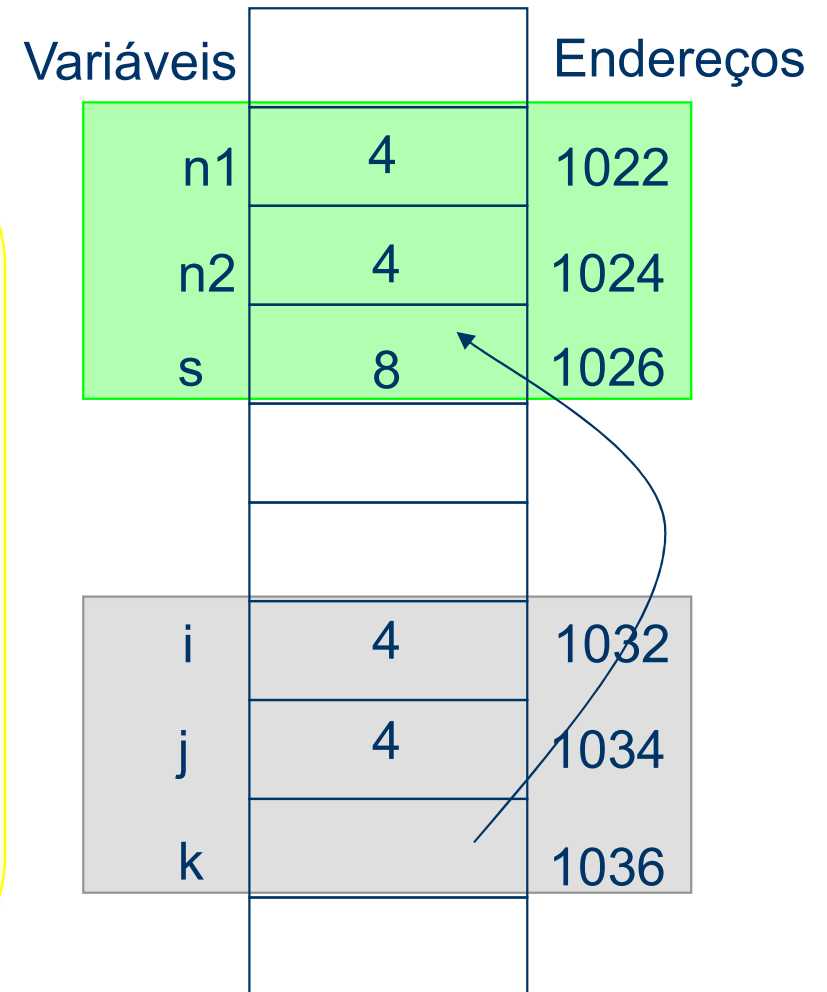
- O uso do (&) indica que o endereço da variável (s) está sendo enviado para a função. Os demais casos são os valores;
- Vamos analisar o comportamento das variáveis na memória.

# Passagem de Parâmetros

- Memória:

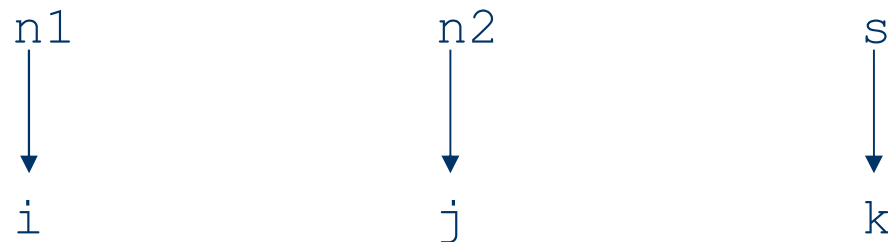
```
#include <stdio.h>
#include <stdlib.h>
void soma (int i, int j, int *k){
    *k = i + j;
}
int main() {
    int n1, n2, s = 0;
    printf("N1 = ");
    scanf("%d", &n1);
    printf("N2 = ");
    scanf("%d", &n2);
    soma(n1, n2, &s);
    printf("Soma = %d\n", s);
    system("PAUSE");
}
```

Variáveis		Endereços
n1	4	1022
n2	4	1024
s	8	1026
i	4	1032
j	4	1034
k		1036



# Passagem de Parâmetros

- Como pode-se observar pelas variáveis, a área verde corresponde ao programa principal e área cinza corresponde à função SOMA;
- Dessa forma, quando a função SOMA é chamada pelo programa principal as variáveis são transferidas para a função:



- Observe que o cálculo da soma é realizado na função e o resultado é devolvido para o programa principal. Isso é possível pelo tipo de passagem de parâmetros.



# Passagem de Parâmetros

- Para a função SOMA existem dois tipos de parâmetros:
  - POR VALOR (i e j): quando a variável é enviada ao programa principal e, qualquer alteração que seja realizada em seu conteúdo, o resultado não é transmitido ao programa principal. Isso pode ser observado na ilustração dos slides anteriores. Nessa apresentação as variáveis são alocadas em endereços distintos;

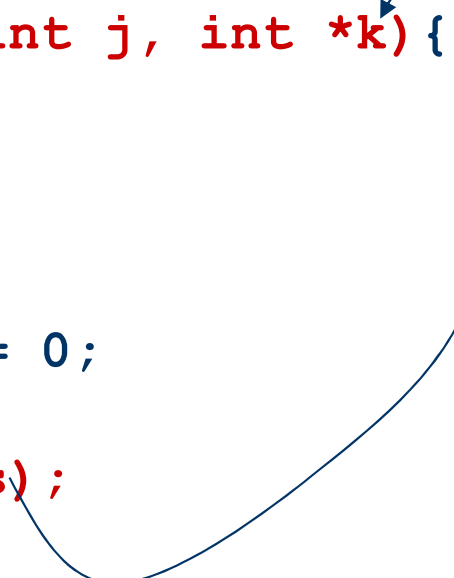
# Passagem de Parâmetros

- Para a função SOMA existem dois tipos de parâmetros:
  - POR REFERÊNCIA (k): quando a variável é enviada ao programa principal e, qualquer alteração que seja realizada em seu conteúdo, o resultado é transmitido ao programa principal. Isso pode ser observado na ilustração dos slides anteriores. Nessa apresentação a variável (k) recebe o endereço da variável (s) e qualquer alteração que seja feita em (k), o resultado é refletido diretamente em (s). Pode-se dizer que (k) aponta para (s);

# Passagem de Parâmetros

- A passagem de parâmetro por referência ocorre da seguinte maneira: o endereço da variável do programa principal, variável *s*, é fornecido para a função por meio do operador (&) e acessado pela variável *k* por meio do operador (\*)

```
void soma (int i, int j, int *k){  
    //...  
}  
  
int main() {  
    int n1, n2, s = 0;  
    //...  
    soma(n1, n2, &s);  
    //...  
}
```



# Tipo de retorno

- Para o exemplo apresentado na função **soma** o retorno definido foi **void**, ou seja, esta função não retorna NADA. Os valores são enviados e devolvidos para o programa principal via passagem de parâmetros;

# Vetor e matriz como parâmetros

- Somente passagem por referência
- Deve-se passar apenas o endereço da posição inicial do vetor ou da matriz
  - Exemplo:  
void soma\_linhas(float m[][5], float v[])

Chamada da função:

```
float mat[8][5], vet[50];  
soma_linhas(mat, vet);
```

## Exercício de fixação (2)

- A prefeitura de uma cidade fez uma pesquisa entre seus habitantes, coletando dados sobre salário, sexo, idade e número de filhos. Crie um registro capaz de armazenar os dados de um habitante e defina um vetor com capacidade para até 500 habitantes. O programa deverá ler os dados de cada habitante e implementar funções para calcular:
  - A média de salários da população;
  - O número médio de filhos;
  - O maior salário e o menor salário;
  - O percentual de mulheres com salário superior a R\$1.500,00.

**Atenção:** Os resultados produzidos por cada função deverão ser impressos pela função **main()** e nunca dentro das funções. **Não use variáveis globais.**

## Exercícios de fixação

- 3- Faça uma função que receba por parâmetro uma *string* S e um caractere C, lidos no programa principal e remove a primeira ocorrência de C em S (se existir). A função deverá alterar a própria *string*, que será impressa depois pelo programa principal. Crie um programa principal que use a função criada.
- 4- Crie uma função que receba como parâmetro um vetor A de 10 números inteiros e substitua todos os valores negativos de A por zero. O vetor resultante será mostrado pelo programa principal.

## Exercícios de fixação (cont.)

5- Faça uma função chamada **calcularSerie()** que calcule e apresente os primeiros 10 termos da série a seguir.

$$\text{Soma} = 100/0! + 99/1! + 98/2! + 97/3! + \dots$$

A função deverá chamar outra função que receba um inteiro  $\geq 0$  como parâmetro e retorne o seu fatorial, para calcular cada termo. Além de apresentar os termos da série, a função deverá ainda retornar a soma total dos termos, que será impressa pelo programa principal.



## Exercícios de fixação (cont.)

6- Um anagrama é uma palavra que é feita a partir da transposição das letras de outra palavra ou frase. Por exemplo, “Iracema” é um anagrama para “America”. Escreva uma função que receba duas *strings* e verifica se a primeira *string* é um anagrama da segunda, ignorando os espaços em branco e acentuação. A função deverá retornar 1, caso seja um anagrama e 0, caso contrário. A mensagem informando se as *strings* formam um anagrama ou não deverá ser impressa pelo programa principal.

OBS: ignore maiúsculas e minúsculas.