centro universitário FEI

CCP130 Desenvolvimento de Algoritmos

Prof. Danilo H. Perico

Modularização de Código: Funções

Funções

- O que são?
  - ***Funções*** são conjuntos de instruções planejadas para cumprir uma tarefa específica

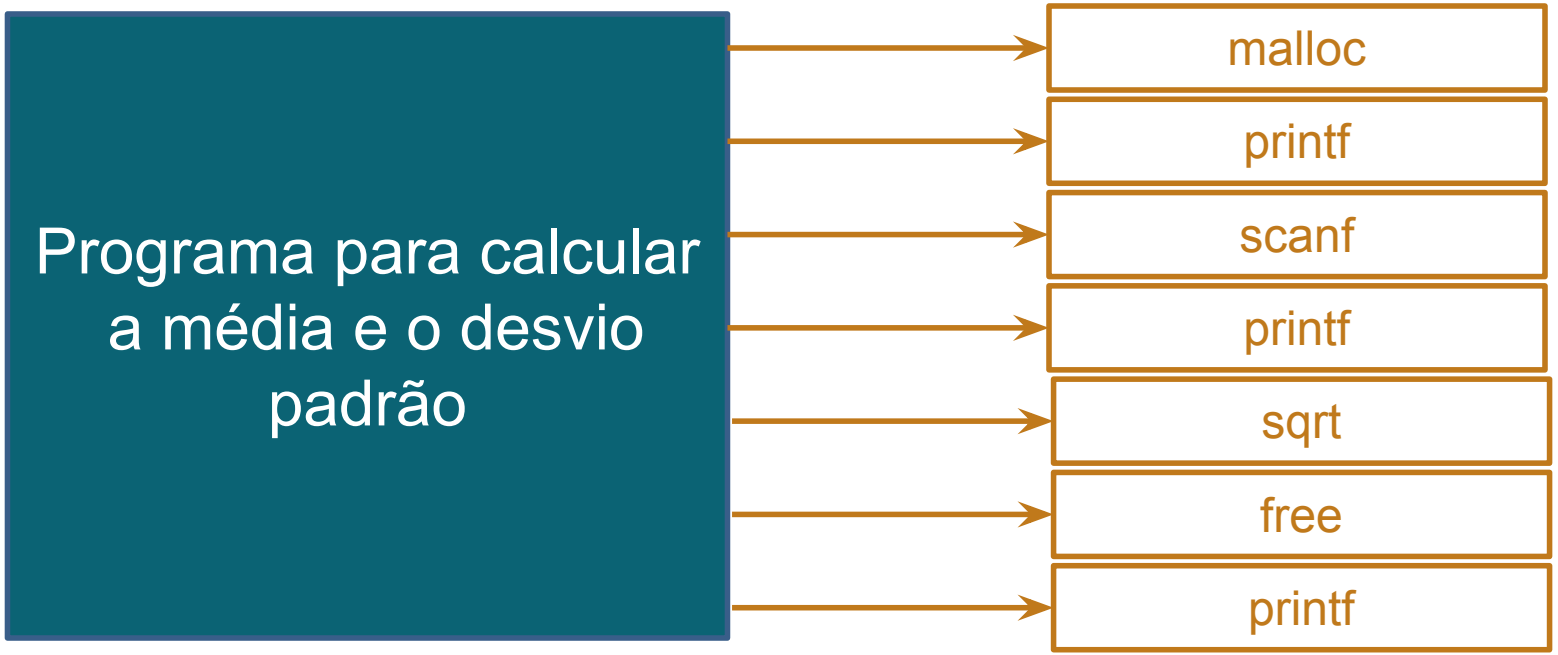
Funções

- Já utilizamos algumas funções:

```
printf("%d", j);
scanf("%f" , &salario);
```

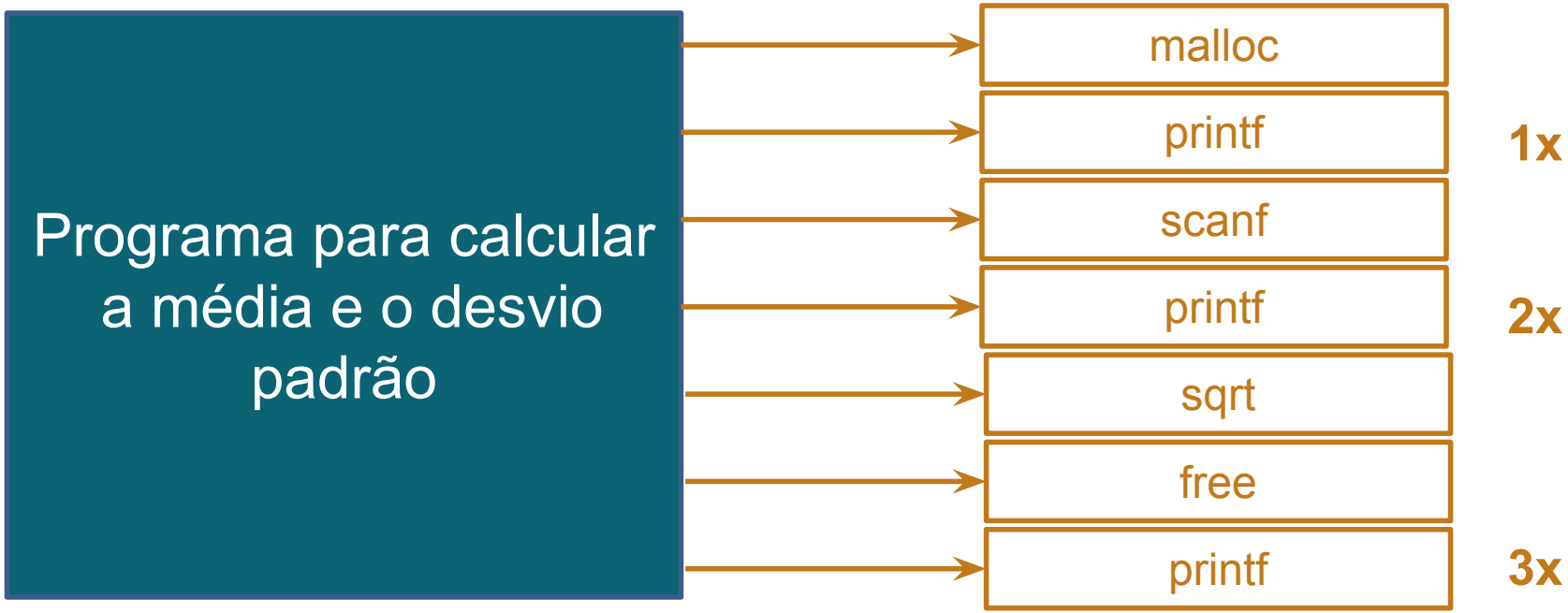
Funções

- Funções dividem **grandes tarefas** em **tarefas menores**



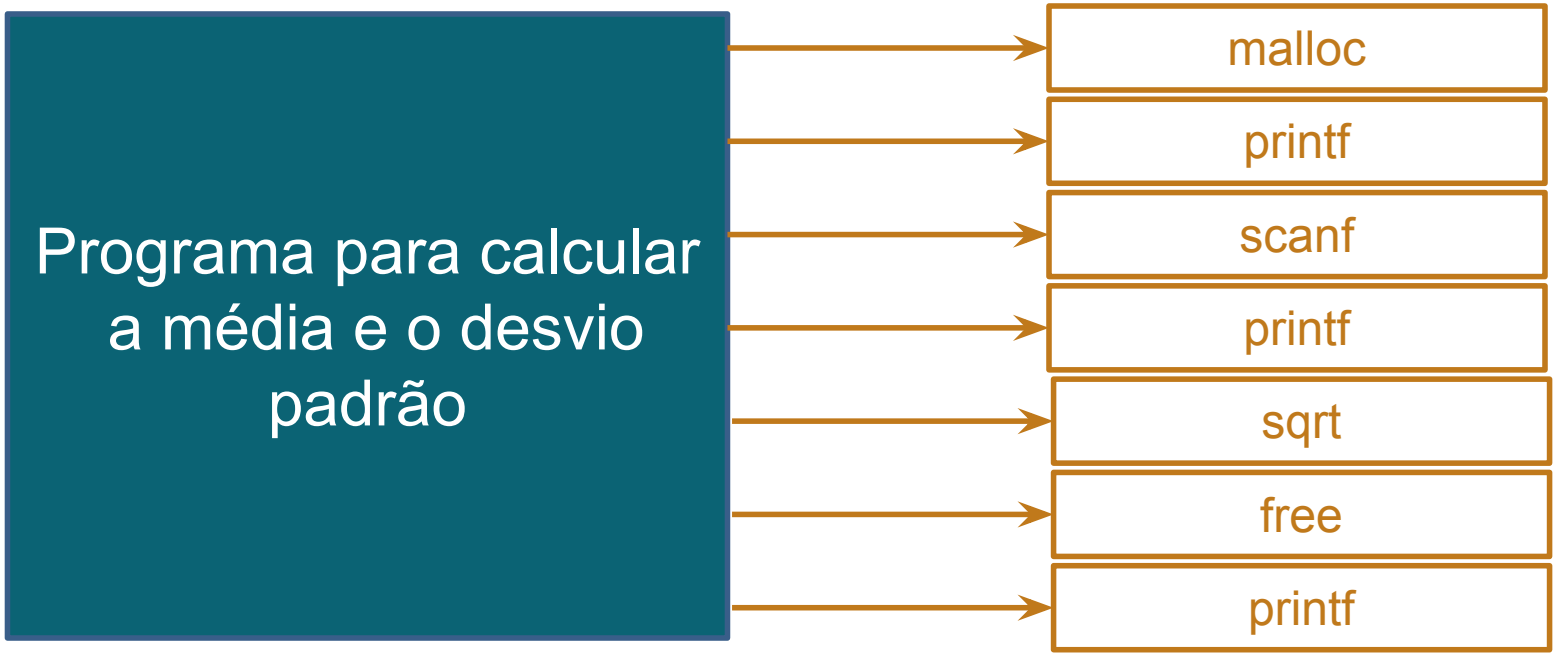
Funções

- Evita que o programador repita o mesmo código várias vezes



Funções

- Promove a manutenibilidade do código
  - Se alguma função mudar, todos os pontos de chamada serão atualizados.



Funções

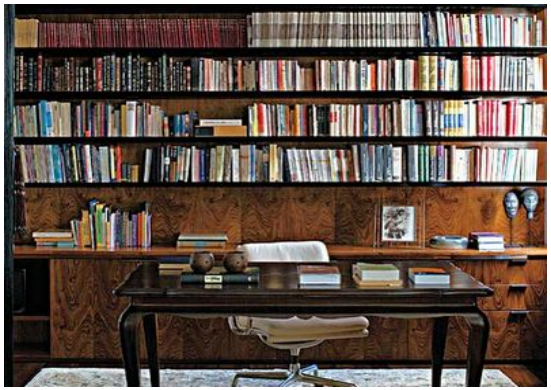
- Invocar uma função pode ser visto como uma contratação de uma pessoa para executar um trabalho específico.
- **Exemplo: função para organizar papéis e livros**

Funções

Contratante
(Programa Principal - ***main***)

Trabalho a fazer
(entrada)

Contratado
(**função**)



Trabalho feito
(saída)

Invocando uma Função

- `printf("%i", j);`
- `scanf("%f" , &salario);`
- `pow( volume, 1/3);`
- `malloc(4);`

- `printf()`, `scanf()`, `pow()`, `malloc()` são funções que foram escritas por outros programadores e são agregadas em nosso programa pelo Linker.

Criando Funções

Sintaxe:

```
tipoRetorno nomeFuncao(tipo1 param1,...,tipoN paramN){

    //Corpo da função

} //Fim da função
```

Criando Funções

Sintaxe:

```
tipoRetorno nomeFuncao(tipo1 param1,...,tipoN paramN){
    //Corpo da função
    return valor;
} //Fim da função
```

- O valor de retorno de uma função é dada pela instrução ***return***

Criando Funções

- Uma função que retorna ***void*** não precisa ter a instrução ***return*** e é chamada de **procedimento**

- **Atenção**: o tipo do retorno de uma função **deve ser o mesmo** que o passado na instrução ***return***

O Comando *return*

- Termina a execução de uma função e **retorna** o valor para o lugar em que a função foi invocada

- Não é permitido o retorno de mais de um valor através do comando ***return***

Onde Criar as Funções?

- As funções definidas pelo programador podem aparecer **antes** ou depois da função ***main***

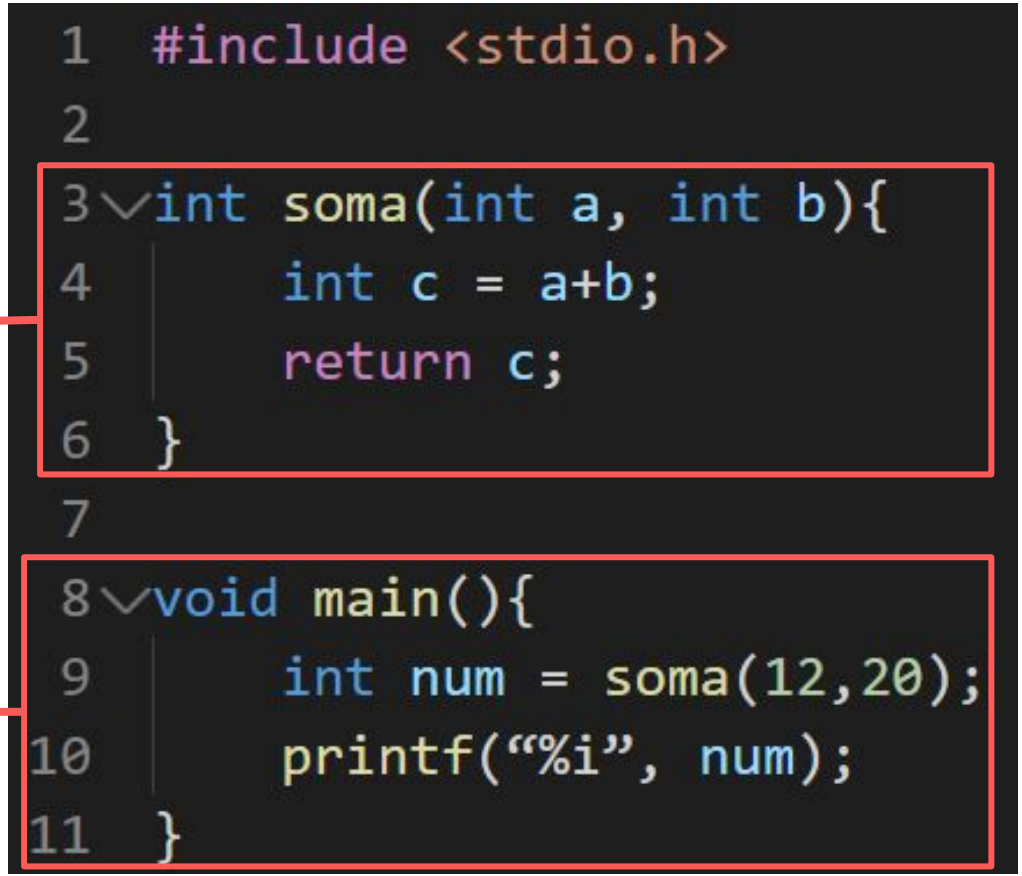
Exemplo - função *soma* definida **antes** de *main*

Função *soma*

retorna um inteiro

Função *main*

retorna nada



```c
#include <stdio.h>

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

void main(){
    int num = soma(12,20);
    printf("%i", num);
}
```

Onde Criar as Funções?

- Caso apareçam **depois** da função ***main*** deve-se declarar seu **protótipo** **antes** da função ***main***

Funções depois da *main*

- **Protótipo de uma função**
  - Identifica a forma como uma função deve ser invocada pelo programador.
  - É justamente seu nome associado aos parâmetros de entrada e aos tipo de retorno.
  - Sabendo de antemão o protótipo o compilador pode verificar se a função é utilizada corretamente.

Exemplo de função após o *main*

Erro!!!

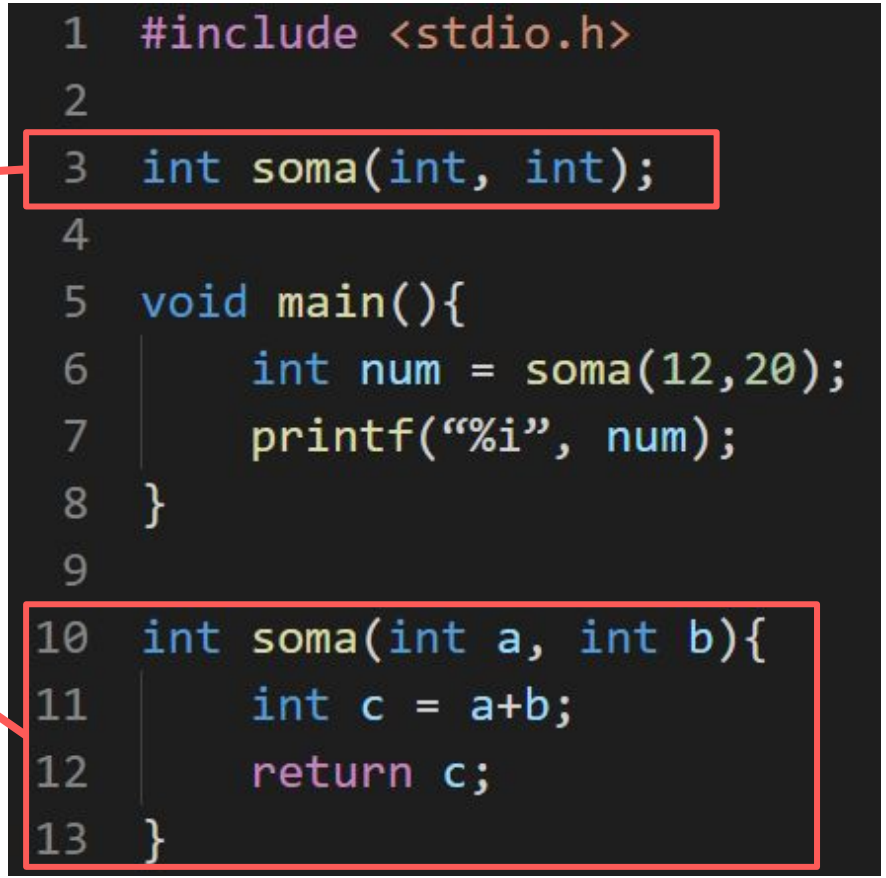
o protótipo não foi declarado antes do *main*

```c
#include <stdio.h>

void main(){
    int num = soma(12,20);
    printf("%i", num);
}

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

Exemplo - protótipo de *soma*

Protótipo:
`int soma(int,int);`

Função criada depois da função *main*



```
1  #include <stdio.h>
2
3  int soma(int, int);
4
5  void main(){
6      int num = soma(12,20);
7      printf("%i", num);
8  }
9
10 int soma(int a, int b){
11     int c = a+b;
12     return c;
13 }
```

Passagem de Parâmetros

Existem **2 tipos** de passagem de **parâmetros:**

- **Por Valor:**
  - Os parâmetros recebem uma **cópia dos valores**
- **Por Referência**
  - Os parâmetros recebem um **endereço de uma variável**

Passagem de Parâmetros

Existem **2 tipos** de passagem de **parâmetros:**

- **Por Valor:**
  - Os parâmetros recebem uma **cópia dos valores**
- **Por Referência**
  - Os parâmetros recebem um **endereço de uma variável**
  - **O C não suporta passagem de valor por referência, porém podemos passar endereços!**

Passagem de Parâmetros - por valor

- Exemplo de passagem de parâmetros por valor

```
1  int soma(int a, int b){
2      int c = a+b;
3      return c;
4  }
5  void main(){
6      int num = soma(12,20);
7      printf("%i", num);
8  }
```

Passagem de Parâmetros - por valor

- Na passagem de parâmetros por **valor,** uma cópia do argumento é feita e é passada para a função

Exemplo - Passagem de Parâmetros - por valor

```c
#include <stdio.h>

void loop_count( int i ) { // uma cópia de i é criada
    printf( "Em loop_count, i = " );
    while( i < 10 )
        printf ( "%d ", i++); // ==> i = 2 3 4 5 6 7 8 9
}

void main( ) {
    int i = 2;
    loop_count( i ); // o argumento i é enviado para loop_count
    printf( "\nEm main, i = %d.\n", i ); // ==> i = 2
}
```

Fonte: Silvana Maria Affonso de Lara <http://wiki.icmc.usp.br/images/6/68/Aula5-Funcoes_2010.pdf>

Passagem de Parâmetros - por referência

- Exemplo de passagem por referência (endereço)

```
1  void troca(int *a, int *b){
2      int c = *a;
3      *a = *b;
4      *b = c;
5  }
6  void main(){
7      int a=2, b=6;
8      int num = troca(&a,&b);
9      printf("%i", num);
10 }
```

ponteiros

Passagem de Parâmetros - por referência

- Na passagem de parâmetros por **referência,** o endereço do argumento é enviado para a função
- Qualquer alteração no valor da variável vai ser percebida globalmente

Passagem de Parâmetros - por referência

- Usar passagem de parâmetros por referência pode ser uma boa estratégia quando a função deveria retornar mais do que um valor (como visto, o ***return*** só permite o retorno de uma valor)
- Vetores são sempre passados por referência

Exemplo - Passagem de Parâm - por referência

```c
#include <stdio.h>

void loop_count( int *i ) { // o endereço é recebido no ponteiro i
    printf( "Em loop_count, i = " );
    while( *i < 10 )
        printf ( "%d ", (*i)++); // ==> i = 2 3 4 5 6 7 8 9
}

void main( ) {
    int i = 2;
    loop_count( &i ); // o endereço de i é enviado para loop_count
    printf( "\nEm main, i = %d.\n", i ); // ==> i = 10
}
```

Fonte: Silvana Maria Affonso de Lara <http://wiki.icmc.usp.br/images/6/68/Aula5-Funcoes_2010.pdf>

Exercícios

Exercícios

1. Implemente uma função ***float potencia(float x, int n)*** que devolva o valor de x^n
2. Escreva uma função que tenha os comprimentos dos dois lados mais curtos de um triângulo retângulo como seus parâmetros. Retorne a hipotenusa do triângulo, calculada usando o teorema de Pitágoras, como o resultado da função. Inclua um programa principal que lê os comprimentos dos lados mais curtos de um triângulo retângulo do usuário e use sua função para calcular o comprimento da hipotenusa. Exiba o resultado. **Utilize a função do exercício anterior**.

Exercícios

3. Crie uma função ***void modifica (int *x)***, que deve mudar o valor passado como parâmetro para o próximo inteiro ímpar. Ex: para a variável inteira *a*, *a*=5; modifica(&*a*); fará com que o valor de *a* se torne 7; *a*=2; modifica(&*a*); fará com que o valor de *a* se torne 3

Exercícios

4. Crie uma função para calcular e retornar o peso de uma pessoa nos outros planetas do Sistema Solar. A função deve ter dois parâmetros: o planeta desejado e o peso em Kg da pessoa na Terra. O programa principal deve receber o peso da pessoa na Terra (em Kg) e o planeta desejado.

   Relação de pesos: 1 Kg na Terra equivale a: 0.37 Kg em Mercúrio; 0.88 Kg em Vênus; 0.38 Kg em Marte; 2.64 Kg em Júpiter; 1.15 Kg em Saturno; 1.17 Kg em Urano; e 1.18 Kg em Netuno.