

Funções em C

Prof^a.Dr^a.Thatyana de Faria Piola Seraphim (ECO)
Prof.Dr.Enzo Seraphim (ECO)

Universidade Federal de Itajubá

thatyana@unifei.edu.br
seraphim@unifei.edu.br

Funções

- ▶ A modularização é um recurso muito importante apresentado nas linguagens de programação, onde um programa pode ser particionado em sub-rotinas específicas.
- ▶ A linguagem C possibilita a modularização por meio das **funções**.
- ▶ Um programa escrito em C, possui no mínimo uma função chamada **main**, por onde a execução do programa começa.
- ▶ Existem muitas outras funções pré-definidas em C, por exemplo: *strcmp*, *strcpy*, entre outras.
 - ▶ Essas funções são inseridas no programa pela diretiva **#include**.
- ▶ O usuário também pode criar quantas funções quiser, dependendo do problema que está sendo resolvido pelo programa.

Definição de Função

Uma função é um conjunto de sentenças que podem ser chamadas de qualquer parte de um programa.

- ▶ As funções **não podem ser aninhadas**, ou seja, uma função não pode ser declarada dentro de outra função.
 - ▶ A razão para isso é permitir um acesso eficiente aos dados.
- ▶ Cada função realiza determinada tarefa, e quando o comando **return** é executado dentro da função
 - ▶ Retorna-se ao ponto em que a função foi chamada pelo programa principal (**main**) ou por uma função principal.

Sintaxe

```
1  tipoRetorno nomeDaFuncao(listaDeParametros){  
2      corpo da funcao  
3      return expressao;  
4  }
```

*int main(int argc, char *argv[]) {*

- ▶ **tipoRetorno**: tipo de valor devolvido pela função.
- ▶ **nomeDaFunção**: identificador ou o nome dado à função.
- ▶ **listaDeParametros**: são as variáveis passadas para a função. Quando a função recebe mais de um parâmetro, esses são separados por vírgula.
- ▶ **expressao**: indica o valor que a função vai devolver para o programa.

Funções

void

Tipo de resultado

Nome da função

Lista de parâmetros

Cabeçalho da função

```
void imprime (int vet[]) {  
    int i;  
    for (i=0; i<10; i++) { printf("%d",  
        vet[i]);  
    }  
}
```

void

```
int soma(int num1, int num2) {  
    int resp;  
    resp = num1 + num2; printf("Soma=%d", resp);  
    * return resp;  
}
```

Declaração de variáveis

Valor devolvido

** return (num1 + num2);*

int
float
char *void*

- ▶ **Tipo de resultado:** o tipo de dado retornado (devolvido) pela função. O tipo sempre aparece antes do nome da função.
- ▶ **Lista de parâmetros:** a lista de parâmetros com tipos com o seguinte formato:

(tipo1 parametro1, tipo2 parametro2, ...)

- ▶ **Corpo da função:** tem que estar definido entre o abre chaves { e o fecha chaves }. **Não há ponto-e-vírgula depois da chave de fechamento.**
- ▶ As constantes, tipos de dados e variáveis declaradas dentro da função são locais à função e não podem ser lidas ou acessadas fora da função.
- ▶ **Valor devolvido pela função:** mediante a palavra **return**, pode-se devolver (retornar) o valor da função.
- ▶ Uma chamada de função produz:
 - ▶ A execução das instruções do corpo da função.
 - ▶ Um retorno para a unidade de programa que fez a chamada depois que a execução da função terminou (ocorre quando a sentença **return** é encontrada).

Nome da Função

- ▶ Um nome de função começa com uma letra ou um (`_`).
- ▶ O nome pode conter tantas letras, números ou (`_`), quanto deseje o programador. Alguns compiladores ignora a partir de uma certa quantidade de caracteres.
- ▶ Letras maiúsculas e minúsculas são distintas para efeito de nome de função.

```
1 int max(int x, int y) //nome da funcao max
2 double media(double x1, double x2) //nome da funcao media
```

Tipo de dado de retorno

- ▶ O usuário deve especificar qual é o tipo de dado que a função vai retornar.
- ▶ O tipo de dado deve ser um dos tipos de C, por exemplo: `int`, `double`, `float` ou `char`; ou então um tipo definido pelo usuário.
- ▶ O tipo **`void`** serve para indicar que a função não retorna nenhum valor.

Exemplos

```
1 int max(int x, int y) //retorna um valor inteiro
2 double media(double x1, double x2) //retorna um double
3 float soma(int numElem) //retorna um float
4 void tela(void) //nao retorna nada
    ( )
```


Resultados de uma função

- ▶ A função pode devolver um único valor, e o resultado é mostrado como uma sentença **return.**
- ▶ O valor retornado por uma função deve seguir as mesmas regras que são aplicadas a um operador de atribuição.
 - ▶ Por exemplo, o valor **int** não pode ser retornado se o tipo de retorno da função for um **char**.
- ▶ Uma função pode ter qualquer número de sentenças **return**.
 - ▶ Sempre que o programa encontra uma instrução **return**, retorna para a instrução que originou a chamada para a função.
 - ▶ A execução de uma chamada à função termina se não encontrar nenhuma instrução **return**; e nesse caso, a execução continua até a chave final do corpo da função.

Chamada a uma função

- ▶ Para que uma função seja executada, é necessário que a função seja **chamada**.
- ▶ Qualquer expressão pode conter uma **chamada a uma função**, a qual redirecionará o controle do programa para a função chamada.
- ▶ Normalmente, a chamada a uma função é realizada pela função **main**, mas a chamada pode ser feita através de outra função.
- ▶ Quando a função termina sua execução, o controle do programa volta para a função **main()** ou para a função de chamada se esta não for o **main**.

Funções

Função para impressão de 10 *

```
1  #include<stdio.h>
2  void desenha(void) {
3      int i; //variavel local
4      for(i=0; i<10; i++){
5          printf("*");
6      } //end for
7  } //end desenha()
8  int main(int argc, char *argv[]){
9      desenha(); //chamada para funcao desenha
10     return 0;
11 } //end main
```

Resultado

Exercício 01

Altere o programa anterior para que seja feita a leitura do número de asteriscos a ser impresso na tela. O número de asteriscos deve ser passado como parâmetro para a função `desenha()`

Funções

Função para impressão de 10 *

```
1  #include<stdio.h>
2  void desenha(int num){
3      int i; //variavel local
4      for(i=0; i<num; i++){
5          printf("*");
6      }//end for
7  }//end desenha()
8  int main(int argc, char *argv[]){
9      int ast;
10     printf("Digite o numero de asteriscos = ");
11     scanf("%d", &ast);
12     desenha(ast); //chamada para funcao desenha
13     → return 0;
14 }//end main
```

Resultado

Digite o numero de asteriscos = 8

Exercício 02

Faça uma função que recebe um número inteiro e **retorna** o quadrado do número inteiro. A função deve ser chamada pelo programa principal (**main()**) que vai **imprimir na tela o valor** do quadrado do número digitado.

Funções

Exercício 02

```
1  #include<stdio.h>
2  int quadrado(int num){
3      int aux;
4      aux = num * num;
5      return aux;
6  }//end quadrado()
7  int main(int argc, char *argv[]){
8      int quad, res;
9      printf("Digite um numero = ");
10     scanf("%d", &quad);
11     res = quadrado(quad); //chamada para funcao quadrado
12     printf("O quadrado de %d= %d", quad, res);
13     printf("O quadrado de 3 = %d", quadrado(3));
14     *return 0;
15 }//end main
```

** printf("%d\n", quadrado(quad));*
for(i=0; i<10; i++)

Exercício 03

Faça uma função que realiza a somatória dos quadrados de números inteiros sucessivos, de zero até um número dado n . A função deve retornar o valor da soma dos quadrados dos n primeiros números. A função deve ser chamada pelo programa principal (**main()**).

Funções

Exercício 03

```
1  #include<stdio.h>
2  int somaQuad(int n){
3      int i, quad, soma = 0;
4      for(i=0; i<n; i++){
5          quad = i * i;
6          soma = soma + quad;
7      } //end for
8      return soma;
9  } //end somaQuad()
10 int main(int argc, char *argv[]){
11     int num, res;
12     printf("Digite um numero = ");
13     scanf("%d", &num);
14     res = somaQuad(num); //chamada para funcao somaQuad
15     printf("Soma = %d \n", res);
16     return 0;
17 } //end main
```

Handwritten annotations in orange:

- Curly braces on lines 2, 4, 5, 6, 7, 8, and 14 are marked with a '3'.
- A circle around the expression `i * i` on line 5 has an arrow pointing to the handwritten text `(i*i)`.
- An arrow points from the handwritten text `pow(i,2);` to the `(i*i)` expression.
- A green comment on line 14 reads: `//chamada para funcao somaQuad`.

Protótipo de Funções

- ▶ Em C é necessário que uma função seja declarada ou definida antes do seu uso.
- ▶ A declaração de uma função sem sua implementação antes do **main** é chamada de **protótipo** da função.
- ▶ O **protótipo** de uma função possui o mesmo cabeçalho da função, com a diferença que os protótipos terminam com ponto-e-vírgula.
- ▶ Um protótipo é composto dos seguintes elementos: tipo, nome da função, parâmetros (que devem estar entre parênteses e é opcional) e um ponto-e-vírgula no final.

Sintaxe do Protótipo

```
tipoRetorno nomeDaFuncao(listaDeParametros);
```

- ▶ Um protótipo declara uma função e passa ao compilador informação suficiente para verificar se a função está sendo chamada corretamente em relação ao número e tipo de parâmetros e ao tipo de retorno da função.
- ▶ Os protótipos são definidos sempre no início do programa, **antes da definição do main()**.
- ▶ O compilador utiliza os protótipos para validar que o número e os tipos de dados dos argumentos na chamada da função, sejam os mesmos que aparecem na declaração formal da função chamada.
- ▶ Caso encontre alguma inconsistência, uma mensagem de erro é visualizada.

- ▶ Uma vez que os protótipos foram processados:
 - ▶ o compilador conhece quais são os tipos de argumentos que ocorreram.
 - ▶ Quando é feita uma chamada para a função, o compilador confirma se o tipo de argumento na chamada da função é o mesmo definido no protótipo.
 - ▶ Se não são os mesmos, o compilador gera uma mensagem de erro.

Exemplo de protótipo

```
1  #include<stdio.h>
2  → int quadrado(int num); //prototipo
3  int main(int argc, char *argv[]){
4      ...
5  }
6  int quadrado(int num){
7      ...
8  }
```

- ▶ A linguagem C permite dois modos para passar parâmetros entre funções.
- ▶ Uma função pode utilizar parâmetros:
 - ▶ Por valor.
 - ▶ Por referência.
 - ▶ Pode não ter parâmetros.

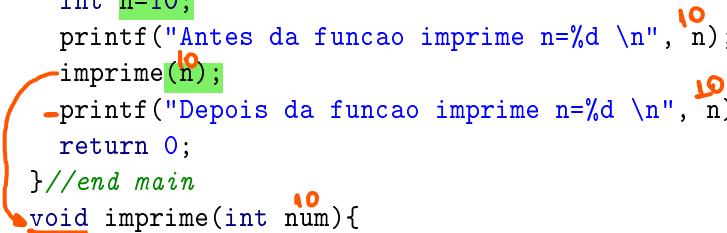
Passagem por Valor

- ▶ É também conhecida como **passagem por cópia**.
- ▶ Quando C compila a função e o código que chama a função:
 - ▶ A função recebe uma cópia dos valores dos parâmetros.
 - ▶ Se o valor de um parâmetro local é mudado, a mudança afeta somente a função e não tem efeito fora dela.

Funções

Exemplo

```
1  #include<stdio.h>
2  void imprime(int num);
3  int main(int argc, char *argv[]){
4      int n=10;
5      printf("Antes da funcao imprime n=%d \n", n);
6      imprime(n);
7      -printf("Depois da funcao imprime n=%d \n", n);
8      return 0;
9  }//end main
10 void imprime(int num){
11     printf("Dentro da funcao num=%d \n", num);
12     num = 200;
13     printf("Dentro da funcao num=%d \n", num);
14 }//end imprime()
```



Resultado do Exemplo

Antes da funcao imprime $n = 10$

Dentro da funcao $\text{num} = 10$

Dentro da funcao $\text{num} = 200$

Depois da funcao imprime $n = 10$;

Passagem de Parâmetro por Referência

- ▶ A passagem de parâmetro por referência é usado quando se deseja que uma função modifique o valor do parâmetro passado e devolva o valor modificado para a função de chamada.
- ▶ O compilador passa o endereço de memória do valor do parâmetro para a função.
 - ▶ Quando se modifica o valor do parâmetro (a variável local), o valor fica armazenado no mesmo endereço de memória.
 - ▶ Ao retornar para a função de chamada, o endereço de memória onde foi armazenado o parâmetro conterà o valor modificado.
- ▶ Para declarar um **parâmetro** como **passagem por referência**, o símbolo **&** deve preceder o nome da variável.

Exercício 04

Faça um programa que leia 2 valores inteiros que serão ordenados de forma crescente. Crie uma função chamada ordena que recebe os 2 valores inteiros e faça a ordenação dos mesmos. Os parâmetros devem ser passados por referência para a função.

Exercício 04

```
1  #include<stdio.h>
2  void ordena(int *a, int *b);
3  int main(int argc, char *argv[]){
4      int a, b;
5      printf("Digite A = ");
6      scanf("%d", &a);
7      printf("Digite B = ");
8      scanf("%d", &b);
9      printf("Antes da ordenacao \n");
10     printf("A = %d, B = %d\n", a, b);
11     ordena(&a, &b);
12     printf("Depois da ordenacao \n");
13     printf("A = %d, B = %d\n", a, b);
14     return 0;
15 } //end main
```

a = 52

b = 25

Continuação do Exercício 04

```
16 void ordena(int *a, int *b){  
17     int n1, n2;  
18     if(5*a < 2*b){  
19         n1 = *a;  
20         n2 = *b;  
21     }//end if  
22     else{  
23         n1 = 2*b;  
24         n2 = 5*a;  
25     }//end else  
26     *a = n1;  
27     *b = n2;  
28 }//end ordena()
```

$n1 =$ 2

$n2 =$ 5

$*a$ 2

$*b$ 5

Diferenças entre os parâmetros por valor e por referência

▶ Parâmetros por valor

- ▶ Os parâmetros por valor são declarados **sem &** e recebem cópias dos valores dos parâmetros passados.
- ▶ A atribuição de uma função a **parâmetros por valor nunca muda o valor do parâmetro original passado.**

▶ Parâmetros por referência

- ▶ Os parâmetros por referência **são declarados com &** e recebem o endereço dos parâmetros passados.
- ▶ As atribuições feitas a parâmetros por referência, **mudam os valores dos parâmetros originais.**