



Disciplina:


Programação Computacional

Prof. Fernando Rodrigues

e-mail: fernandorodrigues@sobral.ufc.br

Aula 13: Programação em C

❖ Funções:

- Forma geral
 - Escopo
 - Tipos
 - Argumentos
 - Chamada por valor e por referência
 - Retornando valores
 - Tipo void
 - Recursividade
- 

A forma geral de uma função

- ▶ A forma geral de uma função em C é:

```
especificador_de_tipo nome_da_função(lista de parâmetros)  
{  
    corpo da função  
}
```

- ▶ As declarações de variáveis são diferentes da declarações de parâmetros

```
f(tipo nomevar1, tipo nomevar2, ..., tipo nomevarN)
```

Regras de escopo de funções

- ▶ As regras de escopo de uma linguagem são as regras que governam se uma porção de código conhece ou tem acesso a outra porção de código ou dados.
- ▶ Em C, cada função é um bloco discreto de código.
- ▶ Variáveis definidas em funções são *locais*.
- ▶ Em C, todas as funções estão no mesmo nível de escopo.

Tipos de funções (conforme o retorno)

- ▶ O primeiro tipo são funções computacionais, que calculam um certo valor e o retornam:
 - ▶ Ex: `sqrt()`, `sin()`, `pow()` etc.
- ▶ O segundo tipo manipula informações e devolve um valor que indica sucesso ou falha.
 - ▶ Ex: `isalpha()`, `islower()`, `isupper()` etc.
- ▶ O último tipo não devolve nenhum valor de retorno explícito (equivalente aos procedimentos).
 - ▶ Ex: `imprimaSoma()` etc

Argumentos de funções

- ▶ Se uma função usa argumentos, ele deve declarar variáveis que aceitem os valores dos argumentos – *parâmetros formais*.

```
is_in(char *s, char c)
{
    while(*s)
        if(*s==c) return 1;
        else s++;
    return 0;
}
```

Argumentos de funções

- ▶ É preciso assegurar que os argumentos usados sejam compatíveis com os tipos de parâmetros.
- ▶ Como no caso das variáveis locais, é possível fazer atribuições a parâmetros formais ou usá-los em qualquer expressão C permitida.
 - ▶ Argumentos declarados nas funções são chamados parâmetros formais, e são as variáveis que receberão os valores passados na chamada da função

Passagem por valor e por referência

- ▶ Há duas maneiras de passar argumentos para funções: *por valor* ou *por referência*.
- ▶ *Passagem por valor*:
 - ▶ copia o valor do argumento no parâmetro formal da função.
- ▶ *Passagem por referência*:
 - ▶ nesse método, o endereço de um argumento é copiado no parâmetro.

Passagem por valor

- ▶ É o padrão para todos os tipos básicos predefinidos (int, char, float e double) e estruturas definidas pelo programador (struct).
- ▶ As alterações nos valores das variáveis usadas para chamar a função não se refletem depois de finalizada a função, ou seja, fora da função.
- ▶ Neste caso a variável real não é alterada, caso se altere o valor do parâmetro formal.

Passagem por valor - Exemplo



Mesmo que o valor de uma variável mude dentro da função, nada acontece com o valor de fora da função.

```
01 include <stdio.h>
02 include <stdlib.h>
03
04 void soma_mais_um(int n){
05     n = n + 1;
06     printf("Dentro da funcao: x = %d\n",n);
07 }
08
09 int main(){
10     int x = 5;
11     printf("Antes da funcao: x = %d\n",x);
12     soma_mais_um(x);
13     printf("Depois da funcao: x = %d\n",x);
14     system("pause");
15     return 0;
16 }
```

Saída

```
Antes da funcao: x = 5
Dentro da funcao: x = 6
Depois da funcao: x = 5
```

Passagem por referência

- ▶ Ponteiros são passados para as funções como qualquer outra variável:
 - Para passar um parâmetro por referência, usa-se o operador “*” na frente do nome do parâmetro durante a declaração da função.
- ▶ Obviamente, é necessário declarar os parâmetros como do tipo ponteiro. E passar uma referência na chamada da função:
 - Na passagem de parâmetros por referência não se passam para a função os valores das variáveis, mas os endereços das variáveis na memória. Na chamada da função, é necessário utilizar o operador “&” na frente do nome da variável que será passada por referência para a função ou utilizar um ponteiro do mesmo tipo de dados.
- ▶ Neste caso, a variável real é alterada, caso se altere o parâmetro formal.

Passagem por referência - Exemplo



A função **scanf()** é um exemplo bastante simples de função que altera o valor de uma variável recebida por parâmetro, e essa mudança se reflete na variável fora da função.

```
01  #include <stdio.h>
02  #include <stdlib.h>
03  int main(){
04      int x = 5;
05      printf("Antes do scanf: x = %d\n",x);
06      printf("Digite um numero: ");
07      scanf("%d",&x);
08      printf("Depois do scanf: x = %d\n",x);
09      system("pause");
10      return 0;
11  }
```

O comando **return**

- ▶ Possui duas funções importantes:
 - ▶ Provoca uma saída imediata da função que o contém.
 - ▶ Pode ser usado para devolver um valor.

Retornando de uma função

- ▶ Existem duas maneiras pelas quais uma função termina a execução e retorna ao código que a chamou:
- ▶ A primeira ocorre quando o último comando da função é executado – ou, conceitualmente, é encontrado o símbolo “}”.
- ▶ A segunda é pelo comando **return**.

Retornando valores

- ▶ Todas as funções, exceto as do tipo **void**, devolvem um valor.
- ▶ Se nenhum comando **return** estiver presente, então o valor de retorno é tecnicamente indefinido.
- ▶ Quando uma função não é declarada como do tipo (de retorno) **void**, ela pode ser um operando em qualquer expressão válida em C.

Retornando valores

- ▶ Permitido:

```
x = power(y);  
if(max(x,y) > 100) printf("greater");  
for(ch=getchar(); isdigit(ch); ) ... ;
```

- ▶ Proibido:




```
swap(x,y) = 100; /* incorrect statement */
```

Retornando valores

- ▶ Embora uma função devolva um valor, você não tem que necessariamente usá-lo.
- ▶ Quando o valor de retorno não é usado, ele é simplesmente **descartado**.
- ▶ Ex. *printf()* devolve um valor, mas, na prática, ninguém o usa.

Funções do tipo **void**

- ▶ Um dos usos de **void** é declarar explicitamente funções que não devolvem valores.
- ▶ Ex.

```
20   
21   
22 
```

```
void quadrado(int n){  
    .....  
    printf("n² = %d ",n*n);  
}
```