

# MC-102 — Aula 10

## Funções

Instituto de Computação – Unicamp

25 de Agosto de 2015

# Roteiro

## 1 Funções

- Definindo uma função
- Invocando uma função
- Exemplos de uso

## 2 O tipo void

## 3 A função **main**

## 4 Exercícios

# Funções

- Um ponto chave na resolução de um problema complexo é conseguir “quebrá-lo” em subproblemas menores.
- Ao criarmos um programa para resolver um problema, é crítico quebrar um código grande em partes menores, fáceis de serem entendidas e administradas.
- Isto é conhecido como modularização, e é empregado em qualquer projeto de engenharia envolvendo a construção de um sistema complexo.

# Funções

## Funções

São estruturas que agrupam um conjunto de comandos, que são executados quando a função é chamada/invocada.

```
scanf("%d", &x);
```

## Funções

As funções podem retornar um valor ao final de sua execução.

```
x = sqrt(4);
```

# Porque utilizar funções?

- Evitar que os blocos do programa fiquem grandes demais e, por consequência, mais difíceis de ler e entender.
- Separar o programa em partes que possam ser logicamente compreendidas de forma isolada.
- Permitir o reaproveitamento de código já construído (por você ou por outros programadores).
- Evitar que um trecho de código seja repetido várias vezes dentro de um mesmo programa, minimizando erros e facilitando alterações.

# Definindo uma função

Uma função é definida da seguinte forma:

```
tipo nome(tipo parâmetro1, ..., tipo parâmetroN) {  
    comandos;  
    return valor de retorno;  
}
```

- Toda função deve ter um tipo (**int**, **char**, **float**, **etc**). Esse tipo determina qual será o tipo de seu valor de retorno.
- Os **parâmetros** são variáveis, que são inicializadas com valores indicados durante a invocação da função.
- O comando **return** devolve para o invocador da função o resultado da execução desta.

## Definindo uma função: Exemplo

A função abaixo recebe como parâmetro dois valores inteiros. A função faz a soma destes valores, e devolve o resultado.

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

- Notem que o valor de retorno (variável **c**) é do mesmo tipo da função.
- Quando o comando **return** é executado, a função para de executar e retorna o valor indicado para quem fez a invocação (ou chamada) da função.

## Definindo uma função: Exemplo

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

- Qualquer função pode invocar esta função, passando como parâmetro dois valores inteiros, que serão atribuídos para as variáveis **a** e **b** respectivamente.

```
int main(){  
    int r;  
    r = soma(12, 90);  
    r = soma (-9, 45);  
}
```

# Definindo uma função: Exemplo

A lista de parâmetros de uma função pode ser vazia.

```
int leNumero () {  
    int c;  
    printf("Digite um número:");  
    scanf("%d", &c);  
    return c;  
}
```

- O retorno será usado pelo invocador da função:

```
int main(){  
    int r;  
    r = leNumero();  
    printf("Numero digitado: %d\n", r);  
}
```

# Definindo uma função

- Funções só podem ser definidas fora de outras funções.
  - ▶ Lembre-se que o corpo do programa principal (**main()**) é uma função.
- Isto está errado:

```
int main(){  
    int f1(int a, int b){  
        return (a+b);  
    }  
    c = f1(9, 90);  
}
```

# Invocando uma função

Uma forma clássica de realizarmos a invocação (ou chamada) de uma função é atribuindo o seu valor à uma variável:

```
x = soma(4, 2);
```

Na verdade, o resultado da chamada de uma função é uma expressão e pode ser usada em qualquer lugar que aceite uma expressão:

## Exemplo

```
printf("Soma de a e b: %d\n", soma(a, b));
```

# Invocando uma função

- Na chamada da função, para cada um dos parâmetros desta, devemos fornecer um valor de mesmo tipo, e na mesma ordem dos parâmetros.
- Ao chamar uma função passando variáveis como parâmetros, estamos usando apenas os seus valores que serão copiados para as variáveis parâmetros da função.
  - ▶ Os valores das variáveis na chamada da função não são afetados por alterações dentro da função.

# Exemplo de função

Veja um exemplo de uso de funções:

```
#include <stdio.h>

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

int main(){
    int res, x1=4, x2=-10;

    res = soma(5,6);
    printf("Primeira soma: %d\n",res);

    res = soma(x1,x2);
    printf("Segunda soma: %d\n",res);
}
```

Obs: Qualquer programa começa executando os comandos da função **main**.

# Exemplo de função

Uma função pode não ter parâmetros, basta não informá-los:

```
#include <stdio.h>

int leNumero(){
    int n;
    printf("Digite um numero:");
    scanf("%d",&n);
    return n;
}

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

int main(){
    int x1, x2;
    x1 = leNumero();
    x2 = leNumero();
    printf("Soma e: %d\n",soma(x1,x2));
}
```

# Exemplo de função

```
#include <stdio.h>

int somaEsquisita (int x, int y) {
    x = x + 1;
    y = y + 1;
    return (x + y);
}

int main () {
    int a, b;

    a=10;
    b=5;
    printf ("Soma de a e b: %d\n", a + b);
    printf ("Soma de x e y: %d\n", somaEsquisita(a, b));
    printf ("a: %d\n", a);
    printf ("b: %d\n", b);
    return 0;
}
```

Os valores de *a* e *b* não são alterados por operações feitas em *x* e *y* !

## Exemplo de função

A expressão contida dentro do comando **return** é chamado de valor de retorno (é a resposta da função). Nada após ele será executado.

```
int leNumero(){
    int n;
    printf("Digite um numero:");
    scanf("%d",&n);
    return n;
    printf("bla bla bla bla");
}

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

int main(){
    int x1, x2;
    x1 = leNumero();
    x2 = leNumero();
    printf("Soma e: %d\n",soma(x1,x2));
}
```

Não imprime *bla bla bla bla!*

# O tipo **void**

- O tipo **void** é um tipo especial.
- Ele representa “nada”, ou seja, uma variável desse tipo armazena conteúdo indeterminado, e uma função desse tipo retorna um conteúdo indeterminado.
- Este tipo é utilizado para indicar que uma função não retorna nenhum valor.

## O tipo **void**

- Por exemplo, a função abaixo imprime o número que for passado para ela como parâmetro:

```
void imprime (int numero) {  
    printf ("Número %d\n", numero);  
}
```

## O tipo **void**

```
#include <stdio.h>

void imprime (int numero) {
    printf ("Número %d\n", numero);
}

int main () {
    imprime (10);
    imprime (20);

    return 0;
}
```

# A função **main**

- O programa principal é uma função especial, que possui um tipo fixo (**int**) e é invocada automaticamente pelo sistema operacional quando este inicia a execução do programa.
- Quando utilizado, o comando **return** informa ao sistema operacional se o programa funcionou corretamente ou não. O padrão é que um programa retorne zero caso tenha funcionado corretamente ou qualquer outro valor caso contrário.

## Exemplo

```
int main() {  
    printf("Hello, World!\n");  
    return 0;  
}
```

# Exercício

- Escreva uma função que computa a potência  $a^b$  para valores  $a$  (double) e  $b$  (int) passados por parâmetro (não use bibliotecas como `math.h`). Sua função deve ter o seguinte protótipo:

**double pot(double a, int b);**

- Use a função anterior e crie um programa que imprima todas as potências:

$$2^0, 2^1, \dots, 2^{10}, 3^0, \dots, 3^{10}, \dots, 10^{10}.$$

# Exercício

- Escreva uma função que computa o fatorial de um número  $n$  passado por parâmetro. Sua função deve ter o seguinte protótipo:

**long fat(long n);** OBS: Caso  $n \leq 0$  seu programa deve retornar 1.

- Use a função anterior e crie um programa que imprima os valores de  $n!$  para  $n = 1, \dots, 20$ .