



## **Aula 10: Funções (Parte III)**

### **Introdução a Programação**

---

**Túlio Toffolo & Puca Huachi**  
<http://www.toffolo.com.br>

BCC201 – 2019/1

Baseado nos slides do Prof. Guillermo Cámara-Chávez

# Aulas anteriores

- Funções
- Ponteiros
- Passagem por referência

# Aula de Hoje

- 1 Exercícios da aula prática
- 2 Exemplos adicionais
- 3 Macros
- 4 Arquivos de cabeçalho
- 5 Biblioteca padrão do C (**libc**)

# Aula de hoje

- 1 Exercícios da aula prática
- 2 Exemplos adicionais
- 3 Macros
- 4 Arquivos de cabeçalho
- 5 Biblioteca padrão do C (**libc**)

# Aula de hoje

- 1 Exercícios da aula prática
- 2 Exemplos adicionais
- 3 Macros
- 4 Arquivos de cabeçalho
- 5 Biblioteca padrão do C (**libc**)

## Exemplo 1

O código C abaixo possui vários **erros**. Identifique estes erros e faça as devidas **correções**:

```
1  int main()
2  {
3      int nro;
4      scanf("%d", nro);
5      duplica(*nro);
6      printf("O dobro do valor digitado é %d\n", &nro);
7      return 0;
8  }
9
10 void duplica(int &nro)
11 {
12     *nro = (*nro) + (*nro);
13 }
```

## Exemplo 1

O código C abaixo possui vários **erros**. Identifique estes erros e faça as devidas **correções**:

```
1  int main()
2  {
3      int nro;
4      scanf("%d", ■ nro);
5      duplica( ■ nro);
6      printf("O dobro do valor digitado é %d\n", ■ nro);
7      return 0;
8  }
9
10 void duplica(int ■ nro)
11 {
12     *nro = (*nro) + (*nro);
13 }
```

## Exemplo 1

O código C abaixo possui vários **erros**. Identifique estes erros e faça as devidas **correções**:

```
1  int main()
2  {
3      int nro;
4      scanf("%d", &nro);
5      duplica(&nro);
6      printf("O dobro do valor digitado é %d\n", nro);
7      return 0;
8  }
9
10 void duplica(int *nro)
11 {
12     *nro = (*nro) + (*nro);
13 }
```



## O que fazer para melhorar o código a seguir?

```
1  int main()
2  {
3      int n1;
4      printf("Digite um nro inteiro positivo: ");
5      scanf("%d", &n1);
6      if (n1 <= 0)
7          printf("Número inválido...\n");
8
9      int n2;
10     printf("Digite um nro inteiro positivo: ");
11     scanf("%d", &n2);
12     if (n2 <= 0)
13         printf("Número inválido...\n");
14
15     int n3;
16     printf("Digite um nro inteiro positivo: ");
17     scanf("%d", &n3);
18     if (n3 <= 0)
19         printf("Número inválido...\n");
20
21     int n4;
22     printf("Digite um nro inteiro positivo: ");
23     scanf("%d", &n4);
24     if (n4 <= 0)
25         printf("Número inválido...\n");
26
27     return 0;
28 }
```

## Exemplo 2

Que tal usar uma função?

```
1  int main()
2  {
3      int n1 = lerNroInteiroPositivo();
4      int n2 = lerNroInteiroPositivo();
5      int n3 = lerNroInteiroPositivo();
6      int n4 = lerNroInteiroPositivo();
7
8      return 0;
9  }
10
11 int lerNroInteiroPositivo()
12 {
13     int n;
14     printf("Digite um nro inteiro positivo: ");
15     scanf("%d", &n);
16     if (n <= 0)
17         printf("Número inválido...\n");
18     return n;
19 }
```

# Aula de hoje

- 1 Exercícios da aula prática
- 2 Exemplos adicionais
- 3 **Macros**
- 4 Arquivos de cabeçalho
- 5 Biblioteca padrão do C (**libc**)

# Macros

- Macros de certa forma lembram funções, mas não são;
- A maior característica de uma macro, é que na declaração utiliza-se a diretiva `#define`
  - O que isso quer dizer?

## Pré-processador e diretivas

- O pré-processador é um programa que examina o código-fonte antes de o mesmo ser compilado;
- As diretivas do pré-processador são recursos que usamos para tornar nossos programas mais claros e fáceis de manter.
- São também sinais para o pré-processador de que algo deve ser alterado no código-fonte antes da compilação.

# Diretivas

## `#include`

- Inclui outro arquivo (geralmente bibliotecas) em nosso código-fonte.
- Na prática, o pré-processador vai substituir a diretiva `#include` pelo conteúdo do arquivo indicado.

# Diretivas

## #define

- Em sua forma mais simples, define constantes simbólicas com nomes mais apropriados.
- Quando um identificador é associado a um `#define`, todas as suas ocorrências no código-fonte são substituídas pelo valor da constante.
- Note que `#define` também pode ser utilizado para criar diretivas mais elaboradas, inclusive aceitando argumentos, chamadas **Macros**.

## Exemplo

```
1 // incluindo a biblioteca stdio
2 #include <stdio.h>
3
4 // definindo o valor de PI
5 #define PI 3.141592
6
7 // definindo o que é um 'beep'
8 // (obs: há formas mais elaboradas de fazer um 'beep')
9 #define BEEP "\x07"
10
11 int main()
12 {
13     printf("pi = %d\n", PI);
14     printf(BEEP);
15     return 0;
16 }
```



# Macros

- Assim como ocorre com a definição de constantes, o pré-processador substitui todas as ocorrências da macro por seu conteúdo
  - Que podem ser instruções em geral;
  - Inclusive outras macros.
- Macros também podem receber argumentos, mas cuidado: há muitas armadilhas.

# Macros

```
1  #include <stdio.h>
2
3  #define SOMA(x,y) x+y;
4
5  int main()
6  {
7      int a, b, resultado;
8      scanf("%d %d", &a, &b);
9
10     resultado = SOMA(a,b);
11     printf("a + b = %d", resultado);
12     return 0;
13 }
```

# Macros

## Erro comum...

```
1  #include <stdio.h>
2
3  #define SOMA(x,y) x+y;
4
5  int main()
6  {
7      int a, b, resultado;
8      scanf("%d %d", &a, &b);
9
10     resultado = 5*SOMA(a,b); // resultado = 5*a+b
11     printf("5 * a + b = %d\n", resultado);
12     return 0;
13 }
```

## Correção do erro anterior...

```
1  #include <stdio.h>
2
3  #define SOMA(x,y) (x+y);
4
5  int main()
6  {
7      int a, b, resultado;
8      scanf("%d %d", &a, &b);
9
10     resultado = 5*SOMA(a,b); // resultado = 5*(a+b)
11     printf("5 * (a + b) = %d\n", resultado);
12     return 0;
13 }
```

# Macros

- Devemos sempre nos lembrar que a macro substituirá trechos dos nossos códigos, e que o tratamento dos argumentos não é o mesmo realizado por uma função.
- Eventualmente, as macros possuem mais que uma linha;
- Neste caso, é necessário indicar o final de cada linha com \

## Resumindo...

- **Macros não são funções.**
- Macros não são reconhecidas por compiladores:
  - São pré-processadas e substituídas no código-fonte.
- Por isso, não há checagem de tipos nos argumentos:
  - O que traz flexibilidade;
  - Mas pode também trazer problemas.
- Em algumas situações, as macros tornam o programa mais rápido, por evitar a leitura de dados da memória e outros procedimentos que ocorrem ao chamarmos uma função.
- **Macros devem ser extremamente pequenas e simples.**

# Aula de hoje

- 1 Exercícios da aula prática
- 2 Exemplos adicionais
- 3 Macros
- 4 Arquivos de cabeçalho
- 5 Biblioteca padrão do C (**libc**)

## Arquivos de cabeçalho (*header files*)

Códigos escritos em C geralmente são divididos em dois arquivos:

- Cabeçalho (arquivo **.h**): contém os protótipos das funções.
- Código fonte (arquivo **.c**): contém a implementação das funções.

Dividir o código em arquivos traz inúmeras vantagens, entre elas:

- Melhor organização;
- Flexibilidade para exportar as funções desenvolvidas por você.



## Exemplo:

Vamos desenvolver uma simples biblioteca contendo funções para converter unidades de temperatura.

- Criaremos dois arquivos: **temperatura.h** e **temperatura.c**

```
_____ temperatura.h _____  
1  double celsiusToFahrenheit(double celsius);  
2  double fahrenheitToCelsius(double fahrenheit);  
3  double celsiusToKelvin(double celsius);  
4  double kelvinToCelsius(double kelvin);  
5  double fahrenheitToKelvin(double fahrenheit);  
6  double kelvinToFahrenheit(double kelvin);
```

## Exemplo:

temperatura.c

```
1  #include "temperatura.h" // inclui o arquivo de cabeçalho
2
3  double celsiusToFahrenheit(double celsius) {
4      return (celsius * 9.0/5.0) + 32;
5  }
6
7  double fahrenheitToCelsius(double fahrenheit) {
8      return (fahrenheit - 32) * 5.0/9.0;
9  }
10
11 double celsiusToKelvin(double celsius) {
12     return celsius + 273.15;
13 }
14
15 double kelvinToCelsius(double kelvin) {
16     return kelvin - 273.15;
17 }
18
19 double fahrenheitToKelvin(double fahrenheit) {
20     return celsiusToKelvin(fahrenheitToCelsius(fahrenheit));
21 }
22
23 double kelvinToFahrenheit(double kelvin) {
24     return celsiusToFahrenheit(kelvinToCelsius(kelvin));
25 }
```

## Exemplo:

Note o compilador recebe como argumento apenas o arquivo com o código fonte.

```
1 $ gcc temperatura.c -Wall -o conversor
```

No entanto, o compilador retornará um erro informando a ausência do método `main`. Se quisermos criar apenas uma biblioteca, fazemos:

```
1 $ gcc -c temperatura.c -Wall -o temperatura.o
```

O comando acima (usando a flag `-c`) compilará o arquivo sem executar a etapa de *link*, gerando o arquivo **temperatura.o** (`.o` de *object file*), que é o binário da biblioteca.

Um executável em C precisa do método `main`. Temos duas opções:

- Adicionar o método `main` em `temperatura.c`
- Criar um outro arquivo com o método `main`.

main.c

```
1  #include <stdio.h>
2  #include "temperatura.h"
3
4  int main()
5  {
6      double celsius;
7      printf("Digite a temperatura em Celsius: ");
8      scanf("%lf", &celsius);
9
10     double fahrenheit = celsiusToFahrenheit(celsius);
11     printf("Temperatura em Fahrenheit: %lf\n", fahrenheit);
12
13     double kelvin = celsiusToKelvin(celsius);
14     printf("Temperatura em Kelvin: %lf\n", kelvin);
15
16     return 0;
17 }
```

## Exemplo:

Para compilar o programa podemos fazer:

```
1 $ gcc main.c temperatura.c -Wall -o conversor
```

Ou, alternativamente, podemos usar o arquivo `.o` gerado anteriormente:

```
1 $ gcc main.c temperatura.o -Wall -o conversor
```

# Aula de hoje

- 1 Exercícios da aula prática
- 2 Exemplos adicionais
- 3 Macros
- 4 Arquivos de cabeçalho
- 5 Biblioteca padrão do C (**libc**)

# Biblioteca padrão do C

- A biblioteca padrão do C (*Standard C Library*, ou **libc**) contém uma série de **funções** e constantes úteis, importadas por meio de diferentes arquivos de cabeçalhos (*header files*).
- Nós já utilizamos algumas (`<stdio.h>` e `<math.h>`).
- Hoje veremos outras funções úteis, presentes em:
  - `<locale.h>`
  - `<limits.h>`
  - `<stdlib.h>`
  - `<time.h>`

## Biblioteca <locale.h>

A linguagem C utiliza o arquivo de cabeçalho `locale.h` para implementar a “localização” de programas (características de idioma ou de uma região).

- Exemplo:

```
1  #include <stdio.h>
2  #include <locale.h>
3
4  int main()
5  {
6      setlocale(LC_ALL, "Portuguese");
7      printf("Utilizando acentuação da língua portuguesa!\n\n");
8      return 0;
9  }
```



## Biblioteca <limits.h>

Esta biblioteca contém constantes para determinar valores máximos e mínimos para vários tipos de dados:

- Exemplo:

```
1 #include <stdio.h>
2 #include <limits.h>
3
4 int main()
5 {
6     printf("int: [%d, %d]\n", INT_MIN, INT_MAX);
7     printf("long: [%ld, %ld]\n\n", LONG_MIN, LONG_MAX);
8     return 0;
9 }
```

## Biblioteca <stdlib.h>

Provê funções para alocação de memória (usaremos muito no futuro), controle de processos, conversão, etc.

```
1  #include <stdio.h>
2  #include <stdlib.h>
3
4  int main()
5  {
6      int i = atoi("-10"); // atoi converte string -> int
7      float f = atof("10.5"); // atof converte string -> float
8      double d = strtod("10.5", NULL); // strtod converte string -> double
9
10     system("clear"); // executa o comando clear no terminal
11
12     srand(0); // seleciona a semente para geração de nros aleatórios
13     int r = rand(); // r recebe um nro aleatório
14     printf("Número aleatório: %d\n\n", r);
15
16     printf("i = %d, f = %f, d = %lf\n", i, f, d);
17     printf("Valor absoluto de i: %d\n\n", abs(i));
18
19     exit(0); // função que finaliza o programa imediatamente
20     return 0;
21 }
```

## Biblioteca <time.h>

Esta biblioteca provê funções para manipulação de data e horários.

- Exemplo:

```
1  #include <stdio.h>
2  #include <time.h>
3
4  int main()
5  {
6      time_t timer = time(NULL);
7      printf("CTIME = %s\n", ctime(&timer));
8      return 0;
9  }
```



Perguntas?