Aula 10: Funções (Parte III)

Introdução a Programação

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

BCC201 – 2019/2

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

Aula de hoje

1. Exercícios da aula prática
2. Exemplos adicionais
3. Macros
4. Arquivos de cabeçalho

Aula de hoje

1. Exercícios da aula prática
2. Exemplos adicionais
3. Macros
4. Arquivos de cabeçalho

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?

```c
int main()
{
    int n1;
    printf("Digite um nro inteiro positivo: ");
    scanf("%d", &n1);
    if (n1 <= 0)
        printf("Número inválido...\n");

    int n2;
    printf("Digite um nro inteiro positivo: ");
    scanf("%d", &n2);
    if (n2 <= 0)
        printf("Número inválido...\n");

    int n3;
    printf("Digite um nro inteiro positivo: ");
    scanf("%d", &n3);
    if (n3 <= 0)
        printf("Número inválido...\n");

    int n4;
    printf("Digite um nro inteiro positivo: ");
    scanf("%d", &n4);
    if (n4 <= 0)
        printf("Número inválido...\n");

    return 0;
}
```

Exemplo 2

Que tal usar uma função?

```c
int main()
{
    int n1 = lerNroInteiroPositivo();
    int n2 = lerNroInteiroPositivo();
    int n3 = lerNroInteiroPositivo();
    int n4 = lerNroInteiroPositivo();

    return 0;
}

int lerNroInteiroPositivo()
{
    int n;
    printf("Digite um nro inteiro positivo: ");
    scanf("%d", &n);
    if (n <= 0)
        printf("Número inválido...\n");
    return n;
}
```

Aula de hoje

1. Exercícios da aula prática
2. Exemplos adicionais
3. Macros
4. Arquivos de cabeçalho

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

```c
// incluindo a biblioteca stdio
#include <stdio.h>

// definindo o valor de PI
#define PI 3.141592

// definindo o que é um 'beep'
// (obs: há formas mais elaboradas de fazer um 'beep')
#define BEEP "\x07"

int main()
{
    printf("pi = %d\n", PI);
    printf(BEEP);
    return 0;
}
```

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

```c
#include <stdio.h>

#define SOMA(x,y) x+y;

int main()
{
    int a, b, resultado;
    scanf("%d %d", &a, &b);

    resultado = SOMA(a,b);
    printf("a + b = %d", resultado);
    return 0;
}
```

Macros

Erro comum...

```c
#include <stdio.h>

#define SOMA(x,y) x+y;

int main()
{
    int a, b, resultado;
    scanf("%d %d", &a, &b);

    resultado = 5*SOMA(a,b); // resultado = 5*a+b
    printf("5 * a + b = %d\n", resultado);
    return 0;
}
```

Macros

Correção do erro anterior...

```c
#include <stdio.h>

#define SOMA(x,y) (x+y);

int main()
{
    int a, b, resultado;
    scanf("%d %d", &a, &b);

    resultado = 5*SOMA(a,b); // resultado = 5*(a+b)
    printf("5 * (a + b) = %d\n", resultado);
    return 0;
}
```

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

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

```c
#include "temperatura.h" // inclui o arquivo de cabeçalho

double celsiusToFahrenheit(double celsius) {
    return (celsius * 9.0/5.0) + 32;
}

double fahrenheitToCelsius(double fahrenheit) {
    return (fahrenheit - 32) * 5.0/9.0;
}

double celsiusToKelvin(double celsius) {
    return celsius + 273.15;
}

double kelvinToCelsius(double kelvin) {
    return kelvin - 273.15;
}

double fahrenheitToKelvin(double fahrenheit) {
    return celsiusToKelvin(fahrenheitToCelsius(fahrenheit));
}

double kelvinToFahrenheit(double kelvin) {
    return celsiusToFahrenheit(kelvinToCelsius(kelvin));
}
```

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

```c
#include <stdio.h>
#include "temperatura.h"

int main()
{
    double celsius;
    printf("Digite a temperatura em Celsius: ");
    scanf("%lf", &celsius);

    double fahrenheit = celsiusToFahrenheit(celsius);
    printf("Temperatura em Fahrenheit: %lf\n", fahrenheit);

    double kelvin = celsiusToKelvin(celsius);
    printf("Temperatura em Kelvin: %lf\n", kelvin);

    return 0;
}
```

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
```

Perguntas?