Prática 3

Introdução à Linguagem C++: Parte 3

3.1 Funções

Neta aula, aprenderemos como escrever e chamar funções em um programa C++. Funções em C++ nos permitem implementar uma das principais formas de abstração em construção de software: *modularização*. Funções são componentes de código com *forte coesão* e *fraco acoplamento*, dois princípios de projeto de software extremamente desejáveis. Há três aspectos fundamentais no estudo de funções: (1) declaração e definição de funções, (2) chamada de funções e (3) passagem de parâmetros por valor e por referência. Os exemplos a seguir ilustram esses aspectos.

3.1.1 Declaração e Definição

O seguinte programa usado em sala para explicar, em detalhes, as formas de declaração, definição e chamada de funções em C++, assim como a forma de passagem de parâmetros por valor.

```
//
// Programa: lb03-01.cpp
//

//
// Exemplo de uma função para calcular o quadrado de um número.
//

#include <iostream>      // cout, endl

//
// Declaração da função square()
//
int square( int );

// Função principal
int main()
{
  //
  // Uso de "using" para evitar escrever o prefixo std::
  //
  using std::cout;
  using std::endl;
```

```
  // Valor a ser elevado ao quadrado
  int a = 10;

  // Escreve o valor a ser elevado ao quadrado e o quadrado.
  cout << a
       << " ao quadrado: "
       << square( a )
       << endl;

  // Indique o término com sucesso do programa.
  return 0;
}
// Fim da função principal


//
// Função para calcular o quadrado de um número inteiro.
//
int square( int x )
{
   return x * x;
}
```

Exercício. Escreva um programa para ler um número real, digamos x, e produzir como saída o cubo deste número. O cálculo do cubo deve ser realizado por uma função, denominada `cubo`, que recebe como parâmetro um único número real e retorna o valor do cubo do número recebido.

3.1.2 Função ou Procedimento?

O seguinte programa ilustra algo bastante confuso das linguagens C e C++: o nome função é empregado para denotar módulos que deveriam ser chamados de procedimentos, pois não retornam valor, mas podem ou não possuir parâmetros de entrada e saída. Em outras situações (que não serão vistas aqui), funções em C/C++ podem possuir mais de um valor de saída (um como valor de retorno e um ou mais através de passagem de parâmetros por referência). Esse tipo de situação não caracteriza, teoricamente, tal "função" em C/C++ como uma função ou um procedimento.

```
//
// Programa: lb03-02.cpp
//

//
// Exemplo de "funções" sem parâmetros de entrada e sem valor de
// retorno.
//

#include <iostream>      // cout, endl

//
// Uso de "using" para evitar escrever o prefixo std::
//
using std::cout;
using std::endl;

// Desta vez, as declarações acima estão fora da função
// principal. Isto se deve ao fato de cout e endl serem usados sem o
// prefixo std:: pelos códigos das funções function1() e function2().

//
// Declaração das de funções sem parâmetros de entrada e sem valor de
// retorno.
```

```
//

// Uma maneira de declarar função sem parâmetros de entrada.
void function1( );

// Outra maneira de declarar função sem parâmetros de entrada.
void function2( void );


// Função principal
int main()
{
  //
  // As sintaxes de chamada das funções são idênticas.
  //

  // Chama função sem parâmetros de entrada.
  function1();

  // Chama função sem parâmetros de entrada.
  function2();

  // Indique o término com sucesso do programa.
  return 0;
}
// Fim da função principal


//
// Definição de uma função que não retorna valor e nem possui
// parâmetros de entrada.
//
void function1( )
{
  cout << "function1 não possui parâmetros de entrada" << endl;

  return;     // o uso de return é opcional aqui
}

//
// Definição de uma função que não retorna valor e nem possui
// parâmetros de entrada.
//
void function2( void )
{
  cout << "function2 também não possui parâmetros de entrada" << endl;

  return;     // o uso de return é opcional aqui
}
```

3.1.3 Passagem de Parâmetros por Referência

O seguinte programa será utilizado para explicar o mecanismo de passagem de parâmetros por referência.

```
//
// Programa: lb03-03.cpp
//

//
// Comparação de passagem de parâmetros por valor e por referência.
//

#include <iostream>      // cout, endl
```

```
//
// Uso de "using" para evitar escrever o prefixo std::
//
using std::cout;
using std::endl;

// Desta   vez,   as   declarações   acima  estão   fora   da   função
// principal. Isto se  deve ao fato de cout e endl  serem usados sem o
// prefixo std:: pelos códigos das funções function1() e function2().

//
// Declaração de uma função para calcular o quadrado de um número.
//
int squareByValue( int );

//
// Declaração de outra função para calcular o quadrado de um número.
//
void squareByReference( int& );


// Função principal
int main()
{
  int x = 2; // Valor a ser elevado ao quadrado por squareByValue()
  int z = 4; // Valor a ser elevado ao quadrado por squareByReference()

  //
  // Uso de squareByValue()
  //
  cout << "x = "
       << x
       << " antes de squareByValue()"
       << endl;

  cout << "Valor retornado por squareByValue(): "
       << squareByValue( x )
       << endl;

  cout << "x = "
       << x
       << " depois de squareByValue()"
       << endl;

  //
  // Uso de squareByReference()
  //
  cout << "z = "
       << z
       << " antes de squareByReference()"
       << endl;

  squareByReference( z );

  cout << "z = "
       << z
       << " depois de squareByReference()"
       << endl;

  // Indique o término com sucesso do programa.
  return 0;
}
// Fim da função principal


// squareByValue() multiplica  o parâmetro de entrada por  ele mesmo e
// retorna o resultado.
int squareByValue( int number )
```

```
{
  return number * number;
}


// squareByReference()  multiplica  o  parâmetro  de entrada  por  ele
// mesmo,  armazena o  resultado no  próprio parâmetro  e  não retorna
// nenhum valor.
void squareByReference( int& numberRef )
{
  numberRef *= numberRef;
}
```

Exercício. Escreva um programa para ler um número real, digamos x, e produzir como saída o cubo deste número. O cálculo do cubo deve ser realizado por uma função, denominada `cubo`, que recebe como parâmetro, por *referência*, um único número real, calcula o cubo deste número, armazena o resultado no próprio parâmetro e não retorna valor (isto é, o tipo de retorno da função é `void`).

3.1.4 Funções com Vários Parâmetros

O seguinte programa contém uma função com mais de um parâmetro de entrada, os quais são passados por valor apenas. A função retorna um valor de saída. Uma função pode ter zero, um ou mais parâmetros e um ou mais desses podem ser passados por referência (e não só por valor).

```
//
// Programa: lb03-04.cpp
//

//
// Exemplo de função inline com mais de um argumento.
//

#include <iostream>      // cin, cout, endl

//
// Declaração de uma função para determinar o menor de dois números.
//
int min( int , int );


// Função principal
int main()
{
  //
  // Uso de "using" para evitar escrever o prefixo std::
  //
  using std::cout;
  using std::cin;
  using std::endl;

  // Valores a serem passados para a função min()
  int a;
  cout << "Entre com um número inteiro: ";
  cin  >> a;

  int b;
  cout << "Entre com outro número inteiro: ";
  cin  >> b;

  //
  // Uso de min()
```

```
  //
  cout << "O menor entre "
       << a
       << " e "
       << b
       << " é "
       << min( a , b )
       << endl;

  // Indique o término com sucesso do programa.
  return 0;
}
// Fim da função principal

// min() retorna o menor de dois números dados.
int min( int x , int y )
{
  return ( x <= y ) ? x : y ;
}
```

Exercício. Escreva um programa para ler dois números reais e escrever o maior deles. O maior dos números deve ser determinado por uma função, chamada *max*, que recebe três parâmetros: dois números passados por valor e uma variável real passada por referência que armazenará o valor do maior dos dois números passados por valor. A sua função não deve retornar nenhum valor (isto é, o tipo de retorno da função é `void`).

3.1.5 Funções `inline`

A linguagem C++ nos permite escrever um tipo especial de função, denominado `inline`. Embora funções `inline` tenham a mesma sintaxe das funções ordinárias (a menos da presença do modificador `inline`), elas são tratadas pelo compilador de maneira completamente distinta das funções ordinárias. Este tratamento diferenciado será explicado, em detalhes, em sala de aula.

```
//
// Programa: lb03-05.cpp
//

//
// Exemplo de função inline
//

#include <iostream>      // cin, cout, endl

//
// Definição de uma função inline para devolver o menor de dois
// números dados.
//
inline int min( const int x , const int y )
{
  return ( x <= y ) ? x : y ;
}


// Função principal
int main()
{
  //
  // Uso de "using" para evitar escrever o prefixo std::
  //
  using std::cout;
  using std::cin;
```

```
  using std::endl;

  // Valores a serem passados para a função min()
  int a;
  cout << "Entre com um número inteiro: ";
  cin  >> a;

  int b;
  cout << "Entre com outro número inteiro: ";
  cin  >> b;

  //
  // Uso de min()
  //
  cout << "O menor entre "
       << a
       << " e "
       << b
       << " é "
       << min( a , b )
       << endl;

  // Indique o término com sucesso do programa.
  return 0;
}
// Fim da função principal
```

Exercício. Escreva um programa para ler dois números reais e escrever o maior deles. O maior dos números deve ser determinado por uma função `inline`, chamada *max*, que recebe dois números reais passados por valor, determina o maior dos dois valores e retorna este valor.