MH1402 Algorithms & Computing II

Lecture 5 Functions (Part 2)

Wu Hongjun

Overview

- Functions
  - Declaration, definition, function call
- Variables
  - Local and Global Variables
- Passing by Reference
- **Function Overloading**
- **Separate Compilation, Header File, Libraries**
- **Math functions in C++ standard library**
- **Recursion**

Function Overloading

- **C++ allows using the same name for different functions , but those functions must have different parameter type lists**
  - **The function being called is the one whose parameter type list matches the type list of invocation**
- **Different parameter type list:**

Examples:

(double, int)	**(double, double)**	**(int, char)**
(double, int, int)	**(double, double, double)**	**(int, int, char)**

The above 6 parameter type lists are different

```
#include <iostream>
using namespace std;

int bar(int, int);
int bar(int, int, int);

int main( )
{
  int x = 2, y = 3, z = 4;

  cout << bar(x, y) << endl;
  cout << bar(x, y, z) << endl;

  return 0;
}
```

```
int bar(int num1, int num2)
{
   return (num1 > num2 ? num1 : num2);
}

int bar(int num1, int num2, int num3)
{
   int num;
   num = (num1 > num2 ? num1 : num2);
   return (num > num3 ? num : num3);
}
```

Example to illustrate Function Overloading

```
#include <iostream>
using namespace std;

int bar(int, int);
double bar(double, double);

int main( )
{
   int x = 2, y = 3;
   double dx = 2.5, dy = 3.5;

   cout << bar(x, y) << endl;
   cout << bar(dx, dy) << endl;

   return 0;
}
```

```
int bar(int num1, int num2)
{
   return (num1 > num2 ? num1 : num2);
}


double bar(double num1, double num2)
{
   return (num1 > num2 ? num1 : num2);
}
```

Another example to illustrate Function Overloading

```
#include <iostream>
using namespace std;

int bar(int, int);
double bar(double, double);

int main() {
  int x = 2, y = 3;
  double dx = 2.5, dy = 3.5;

  cout << bar(x, y) << endl;
  cout << bar(x, dy) << endl;

  return 0;
}
```

```
int bar(int num1, int num2)
{
   return (num1 > num2 ? num1 : num2);
}

double bar(double num1, double num2)
{
   return (num1 > num2 ? num1 : num2);
}
```

Example to illustrate Function Overloading:
Type mismatching in function call

Error: Call of overloaded function is ambiguous

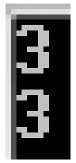
```
#include <iostream>
using namespace std;

int bar(int, int, int);
double bar(double, double, double);

int main() {
  int x = 2, y = 3, z = 4;
  double dx = 2.5, dy = 3.5,
       dz = 4.5;

  cout << bar(x, y, z) << endl;
  cout << bar(dx, y, dz) << endl;

  return 0;
}
```

```
int bar(int num1, int num2, int num3)
{
   int num;
   num = (num1 > num2 ? num1 : num2);
   return (num > num3 ? num : num3);
}

double bar(double num1, double num2, double
num3)
{
   double num;
   num = (num1 > num2 ? num1 : num2);
   return (num > num3 ? num : num3);
}
```

Another example to illustrate function overloading: Type mismatching in function call

Error: Call of overloaded function is ambiguous

```
#include <iostream>
using namespace std;

int bar(int, int);

int main( )
{
   int x = 2, y = 3;
   double dx = 2.5, dy = 3.5;

   cout << bar(x, y) << endl;
   cout << bar(dx, dy) << endl;

   return 0;
}

int bar(int num1, int num2)
{
    return (num1 > num2 ? num1 : num2);
}
```



```
3
3
```

Another example of type mismatching in function call

No Compilation Error

Reason: There is no function overloading, so there is no ambiguity in deciding which function to use: only one function with name bar( )

```
#include <iostream>
using namespace std;

int bar(int, int);

int main()
{
  int x = 2, y = 3;
  double dx = 2.5, dy = 3.5;

  cout << bar(x, y) << endl;
  cout << bar(dx, dy, 4.5) << endl;

  return 0;
}

int bar(int num1, int num2)
{
   return (num1 > num2 ? num1 : num2);
}
```

Another example of type mismatching in function call

Compilation Error

Reason: the number of parameters of function bar does not match the number of arguments

Separate Compilation

Where to define functions?

- **Method 1: All the functions (including the main function) are defined in a single .cpp file**
  - **Not easy to write and read when the code size is huge**
  - **It is desirable to divide a large file into small files**
- **Method 2: Define functions in multiple .cpp files**
  - **Some functions (closely related) are defined in a single .cpp file**
  - **Most commonly used in developing C++ software (recommended!!!)**
- **Method 3: Define functions in header files (.h)**
  - **Note that the main function should not be defined in a header file**
  - **Sometimes this method is used**

Separate Compilation

- **When we define functions in different .cpp files, we should ensure that in every .cpp file, each function is declared (or defined) before being called.**
- **Each .cpp file gets compiled separately into a .o file (object file). Then the linker (in the compiler) links the .o files into an executable program**

Separate Compilation: Advantages

- **Advantage 1. Compile functions separately allows the functions being tested separately before the code(s) that call them are written**
- **Advantage 2. Easy to replace one module with another equivalent module**
  (Example: Patching/Fixing software bugs in a module, common in software industry.)
- **Advantage 3. Allow the functions being distributed without disclosing the source codes**
  (important for the commercial software)

Separate Compilation: Example

must be declared

bar.cpp

```
#include <iostream>   //cout in iosteam
#include <cmath>      //sqrt, pow in cmath

using namespace std;

double func(double x, double y)
{
   double z;
   //square root of x plus square of y
   z =  sqrt(x) + pow(y,2);
   cout << "Test separate compilation" << endl;

   return z;
}
```

foo.cpp

```
#include <iostream>

using namespace std;

double func(double,double);

int main( )
{
   double da = 4.1, db = 3.5, dc;
   dc = func(da, db);
   cout << dc << endl;

   return 0;
}
```

Separate Compilation

the output file name is test.exe

- How to compile foo.cpp and bar.cpp in the previous slides ?
  - Method1: In DOS command prompt, we can compile them using C++ compiler g++

  (Refer to Lab 2 on how to compile from DOS command prompt)

```
g++ -c foo.cpp                 // compile foo.cpp into foo.o file
g++ -c bar.cpp                 // compile bar.cpp into bar.o file
g++ -o test  foo.o  bar.o      // link and generate test.exe
```

```
(or simply:  g++ -o test foo.cpp bar.cpp )
```

(compile in DOS command prompt will not be tested. But it is useful.)

Separate Compilation

- **How to compile foo.cpp and bar.cpp in the previous slides ?**
  - **Method 2: Using CodeBlocks (Lab 4 gives more details):**
  - **We create an empty project (say, with name separate_compilation), then add these two .cpp files into this project.**
  - **Now you build this project and get an executable program separate_compilation.exe**
  - **When we build in CodeBlocks, the CodeBlocks executes those three commands (given in the previous slide) for us automatically.**

Separate Compilation

Warning: do NOT use *#include <xxx.cpp>* in your program!
Bad programming style!

Header file

- **With extension .h (or no extension)**
- **A main purpose of a header file is to contain function declarations for other files to use**

Header file: Example 1

```
#include <iostream>

using namespace std;

int main( )
{
   cout << "Hello, world!" << endl;
   return 0;
}
```

cout **is declared in the header file** ***iostream***

When we use ***#include <iostream>,*** **we're requesting that all of the content from the header file named "*****iostream*****" be copied into the including file.**

Header file: Example 2 (write your own header file)

bar.cpp

```
#include <iostream>
#include <cmath>

using namespace std;

double func(double x, double y)
{
   double z;
   z =  sqrt(x) + pow(y,2);

   return z;
}
```

bar.h

```
double func(double, double);
```

foo.cpp

```
#include <iostream>
#include "bar.h"

using namespace std;

int main( )
{
   double da = 4.1, db = 3.5, dc;
   dc = func(da, db);
   cout << dc << endl;

   return 0;
}
```

Use separate compilation method to compile these three files.

When the compiler compiles the #include "bar.h" line, it copies the contents of *bar.h* into the current file at that point, so we have the function declaration in foo.cpp.

Header file

- Use angled brackets to include header files that come with the compiler.

  Examples: #include <iostream>

  #include <cmath>

- Use double quotes to include any other header files.

  Example: #include "bar.h"

- To know more information of header file, such as the header guard, you may read this webpage:

  http://www.learncpp.com/cpp-tutorial/19-header-files/

Library

- **The Standard C++ library (std)**
    - **It is a collection of pre-defined functions and other program elements (classes)**
    - **Those functions are precompiled into library**
        - **You may roughly view a library as a collection of .o (object) files**

Library

- **How to use the standard C++ library?**

  For example:

  - **Function sqrt( ) is defined in the standard C++ library;**
  - **Function sqrt( ) and other maths functions are declared in the header file cmath.h ;**
  - **When we have #include <cmath> in a .cpp file, it effectively inserts the declaration of maths functions into this .cpp file before compilation.**
  - **It means that sqrt( ) is declared in this .cpp file, and we can use the function sqrt( ) in this .cpp file. (Note that in a C++ file, a function must be declared or defined in that .cpp file before being used.)**

Library

- **Two main types of libraries (in binary format)**

  http://en.wikipedia.org/wiki/Library_(computing)

  - **Static libraries**
    - **The code of the function being called is copied into the executable program by linker**
    - **Usually with a file name extension of .a (unix) or .lib (windows)**
  - **Shared libraries**
    - **The code of the function being called is loaded from individual shared library into memory at load time or run time (so the size of the executable program is reduced)**
    - **Usually with file name extension of .so or .dll (windows)**
- **A user can create his/her own libraries**

Math functions in C++

- **How to perform exponentiation, logarithm … in C++?**
  - **The cmath library declares a number of math functions**
    http://www.cplusplus.com/reference/cmath/
- **Typically, the input parameter to the math function is double type, and the returned value is also double type (there are only a few exceptions)**
  - **If the input type is not double, the input value is converted to double, then used by the function**

- An example of computing square root using the function sqrt

```
#include <cmath>       // cmath must be included here.
                       // sqrt is declared in the header file cmath
#include <iostream>
using namespace std;

int main()
{
      int x = 3;
      cout << sqrt(x);    //  the input to sqrt is converted to double
                          //  the function sqrt returns a double
      return 0;
 }
```

Math functions in C++

Euler's constant,
Euler's number

- **Some math functions declared in cmath:**

`sqrt(x)`	`square root of x`
`abs(x)`	`absolute value of x (type different in C)`
`pow(x, p)`	`x to the power p, pow(2,3) returns 8.0`
`exp(x)`	`e to the power x,e is constant 2.71828…`
`log(x)`	`natural logarithm of x (base e)`
`log10(x)`	`logarithm of x (base 10)`
`ceil(x)`	`ceiling of x (rounds up)`
`floor(x)`	`floor of x (rounds down)`

Math functions in C++

- **Some math functions declared in cmath: (cont.)**

`sin(x)`	`sine of x`
`cos(x)`	`cosine of x`
`tan(x)`	`tangent of x`
`asin(x)`	`inverse sine of x`
`acos(x)`	`inverse cosine of x`
`acos(x)`	`inverse tangent of x`

`Note that radian is used in the above functions`

Math functions in C++

- **Caution: math function typically performs computation on data in double type, so round-off error may occur.**
- **Example:**

```
#include <cmath>
// ............
int x = pow(10,2)*1;
cout << x;    // the output may be 99 on some computer
```

The reason is that the input parameters and output (returned) value of pow in cmath is double. There may be round-off error in the computation involving double (floating-point). For example, the value of pow(10,2)*1 may be slightly less than 100 (the difference is negligibly small). However, when we assign this value to int x, the fractional part get discarded, so the integer 99 is assigned to x.

Math functions in C++

- **Caution: many math function returns a double type data.**
- **Example:**

There is compilation error in the statement pow(2,3) % 2;

The reason is that the output of pow (the returned value of pow) is double. However, the modulo operator % requires two integers, i.e., for x % y, both x and y should be integer. So there is compilation error.

Recursion

Recursion

- **A function can call itself for repeating similar computing**
  - **Alternatively, the same task can be achieved using loop**
- **Example: integer factorial**

```
#include <iostream>

using namespace std;

unsigned long long factorial(unsigned int n)
{
   if (n == 0)
      return 1;
   else
      return n * factorial(n - 1);
}
```

```
//compute the factorial of
//a small integer
int main( )
{
   unsigned int num = 4;
   unsigned long long result;
   result = factorial(num);
   cout << result << endl;
}
```

Recursion

- **In the previous slide,**
  we first call function factorial(4),
  then inside function(4), factorial(3) is called;
  then inside function(3), factorial(2) is called;
  then inside function(2), factorial(1) is called;
  then inside function(1), factorial(0) is called.
  Now factorial(0) returns 1 to function factorial(1);
  then factorial(1) returns 1*1 to function factorial(2);
  then factorial(2) returns 2*1*1 to function factorial(3);
  then factorial(3) returns 3*2*1*1 to function factorial(4);
  then factorial(4) computes and returns 4*3*2*1*1 (done)