1. 普通非模板类或函数声明为友元。这样声明的友元类或友元函数可以访问类模板实例化的任意实例的private和protected成员。

bar.h

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\* 文件名 ：bar.h

/\* 作者 ：刘珅珅

/\* 日期 ：2011/11/8

/\* 功能描述 ：类模板中的友元，非模板类或普通函数为友元

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#ifndef BAR\_H

#define BAR\_H

#include <iostream>

using namespace std;

class FoolBar;

void func();

template <class T> class Bar

{

public:

friend class FoolBar;

friend void func();

Bar();

private:

int num;

void fcn();

};

#endif

foolbar.h

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\* 文件名 ：foolbar.h

/\* 作者 ：刘珅珅

/\* 日期 ：2011/11/8

/\* 功能描述 ：类模板中的友元类

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#ifndef FOOLBAR\_H

#define FOOLBAR\_H

#include <iostream>

using namespace std;

class FoolBar

{

public:

FoolBar();

};

#endif

bar.cpp

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\* 文件名 ：bar.cpp

/\* 作者 ：刘珅珅

/\* 日期 ：2011/11/8

/\* 功能描述 ：类模板中的友元 \*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#include "bar.h"

#include <iostream>

using namespace std;

template <class T> Bar <T> :: Bar ()

{

cout << "类模板的构造函数" << endl;

}

template <class T> void Bar <T> :: fcn()

{

cout << "模板类的私有函数" << endl;

}

foolbar.cpp

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\* 文件名 ：foolbar.cpp

/\* 作者 ：刘珅珅

/\* 日期 ：2011/11/8

/\* 功能描述 ：类模板中的友元类

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#include "foolbar.h"

#include <iostream>

using namespace std;

FoolBar :: FoolBar()

{

cout << "友类的构造函数" << endl;

}

test.cpp

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\* 文件名 ：test.cpp

/\* 作者 ：刘珅珅

/\* 日期 ：2011/11/8

/\* 功能描述 ：类模板中的友元的测试 \*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#include "bar.cpp" // 这里要包含.cpp文件，这是由于编译模板时，需要定义模板的源代码

#include <iostream>

using namespace std;

// 友元函数func()不能定义在bar.h或bar.cpp中，否则会引起重复包含

void func()

{

Bar <int> bar; //类的友元函数访问类的private和protected成员时，需要使用类对象进行访问

bar.num = 0;

bar.fcn();

cout << "友元函数" << bar.num << endl;

}

int main(int argc, char \*argv[])

{

Bar <int> bar;

func();

return 0;

}

类FoolBar和fcn函数可以访问Bar类的任意实例的private成员和protected成员。

编写类模板时，最好将类模板的定义及其成员函数的定义都写在同一个头文件中。

1. 友元可以是类模板或函数模板，最常见的一种方式如下所示：

list.h

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\* 文件名 ：list.h

/\* 作者 ：刘珅珅

/\* 日期 ：/11/7

/\* 功能描述 ：定义类模板 \*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#ifndef LIST\_H

#define LIST\_H

#include <iostream>

using namespace std;

template <class T> class List; // 类模板的声明

template <class T> std :: ostream& operator << (std :: ostream&, const List <T>&); // 函数模板的声明

// 类模板的定义

template <class T> class Node

{

public:

friend class List <T>;

//这种声明友元函数的形式只能在vs2008及以上版本中使用，不能在VC6.0中使用

friend std :: ostream& operator << <T> (std :: ostream&, const List <T>&);

Node() : pNext(0), pT(0) {}

private:

Node \*pNext; //在类本身的作用域内部，使用类模板的名字时可以不指定形参

T \*pT;

};

template <class T> class List

{

public:

friend std :: ostream& operator<< <T> (std :: ostream&, const List <T>&);

List();

void Add(T&);

void Remove(T&);

T\* Find(T&);

private:

Node <T> \*pFirst; // 注意：使用类模板的名字时，必须指定类型形参

};

template <class T> List <T> :: List()

{

pFirst = 0;

}

//添加链表节点

template <class T> void List <T> :: Add(T& t)

{

Node <T> \*temp = new Node <T>;

temp->pT = &t;

temp->pNext = pFirst;

pFirst = temp;

}

//删除链表节点

template <class T> void List <T> :: Remove(T& t)

{

Node <T> \*q = 0;

if (\*(pFirst ->pT) == t)

{

q = pFirst;

pFirst = pFirst->pNext;

}

else

{

for (Node <T> \*p = pFirst; p->pNext; p = p->pNext)

{

if (\*(p->pNext->pT) == t)

{

q= p->pNext;

p->pNext = q->pNext;

break;

}

}

}

if(q)

{

delete q->pT;

delete q;

}

}

template <class T> T\* List <T> :: Find(T& t)

{

for (Node <T> \*p = pFirst; p; p = p->pNext)

{

if (\*(p->pT) == t)

{

return p->pT;

}

}

return 0;

}

template <class T> std :: ostream& operator << (std :: ostream &os, const List <T> &q)

{

os << "重载输出操作符" << endl;

for(Node <T> \*p = q.pFirst; p; p = p->pNext)

{

os << \*(p->pT) << " ";

}

return os;

}

#endif

test.cpp

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\* 文件名 ：test.cpp

/\* 作者 ：刘珅珅

/\* 日期 ：/11/7

/\* 功能描述 ：类模板的测试 \*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#include "list.h"

int main(int argc, char \*argv[])

{

List <float> floatList;

for (int i = 1; i < 7; ++i)

{

floatList.Add( \*(new float(i + 0.6)));

}

// 这里有4个输出操作符，只有第3个输出操作符是重载的

cout << "输出链表的所有内容" << endl << floatList << endl;

float b = 3.6;

float \*pa = floatList.Find(b);

if (pa)

{

floatList.Remove(\*pa);

}

cout << floatList << endl;

//让输出不一闪而逝

system("pause");

return 0;

}