1，友元函数的定义和作用

我们已知道类具有封装和信息隐藏的特性。只有类的成员函数才能访问类的私有成员，程序中的其他函数是无法访问私有成员的。非成员函数可以访问类中的公有成员，但是如果将数据成员都定义为公有的，这又破坏了隐藏的特性。另外，应该看到在某些情况下，特别是在对某些成员函数多次调用时，由于参数传递，类型检查和安全性检查等都需要时间开销，而影响程序的运行效率。

为了解决上述问题，提出一种使用友元的方案。友元是一种定义在类外部的普通函数，但它需要在类体内进行说明，为了与该类的成员函数加以区别，在说明时前面加以关键字friend。友元不是成员函数，但是它可以访问类中的私有成员。友元的作用在于提高程序的运行效率（即减少了类型检查和安全性检查等都需要的时间开销），但是，它破坏了类的封装性和隐藏性，使得非成员函数可以访问类的私有成员。

2，上述的访问不是直接访问，对于普通私有成员变量是通过对象访问，对于私有静态变量是通过类访问。

为什么不能直接访问呢？

先了解一下为什么成员函数可以直接访问成员变量？

成员函数能够访问类的成员变量是因为传递了指向当前对象的this指针，它如果访问数据成员对其操作是this指向的对象的数据成员，是有实际意义的 。

友元函数不是成员函数，没有传递隐藏的this指针，只能间接访问。

这点其实和静态成员函数一样，静态成员函数也是没有this指针的，所以它只能访问静态成员变量或者通过对象访问非静态成员变量。

例子：

class Rect

{

public:

Rect() *// 构造函数，计数器加1*

{

count++;

}

*//Rect(const Rect& r)*

*//{*

*// width = r.width;*

*// height = r.height;*

*// count++;*

*//}*

~Rect() *// 析构函数，计数器减1*

{

count--;

}

static int getCount() *// 返回计数器的值*

{

return count;

}

friend int get();

private:

int width;

int height;

static int count; *// 一静态成员做为计数器*

};

int Rect::count = 0; *// 初始化计数器*

int get()

{

return Rect::count;*//友元函数通过类访问私有静态成员变量*

}

int main()

{

Rect rect1;

cout<<"The count of Rect: "<<Rect::getCount()<<endl;*//通过类访问公有静态成员函数，输出1*

Rect rect2(rect1); *// 使用rect1复制rect2，此时应该有两个对象*

cout<<"The count of Rect: "<<Rect::getCount()<<endl; *//输出1*

cout << get() << endl;*//输出1*

*//cout << Rect::count << endl;//不能编译通过，不能访问私有成员*

system("pause");

return 0;

}

3，类和类之间的友元关系不能继承。

下边转载自：

http://blog.csdn.net/shandianling/article/details/7469361

C++ Primer中有如下描述：**友元关系不能继承。基类的友元对派生类的成员没有特殊访问  
权限。如果基类被授予友元关系，则只有基类具有特殊访问权限，该基类的派生类不能访问授予友元关系的类。**

 然而通过实践发现，VS编译器并没有安装上述描述来处理，下面的规则与上述描述相悖，却符合VS编译器的处理规则。

注：有待通过g++编译器来验证。

**1 友元类的继承问题**

1.1 **A类的友元B的派生类C 不能访问A类的private或protect成员变量。但可以通过B提供的接口来访问A。(废话肯定可以)**

**[cpp]**  [***view plain***](http://www.voidcn.com/article/p-ktcxihws-bgr.html)  [***copy***](http://www.voidcn.com/article/p-ktcxihws-bgr.html)

1. #include <iostream>
2. **using** **namespace**   std;
3. **class** B;
4. **class** A
5. {
6. **int** a;
7. **public**:
8. A(**int** x=0) { a=x; }
9. **friend** **class** B;
10. };
11. **class** B
12. {
13. **int** b;
14. **public**:
15. **void** fun(A& ob){ cout << ob.a << endl;}
16. };
18. **class** C:**public** B
19. {
20. **public**:
21. //void fun2(A& ob){ cout <<ob.a <<endl;}   //派生类新加的函数却不能访问A，此句会报错
22. };
24. **void**   main()
25. {
26. A a(55);
27. C c;
28. c.fun(a); //C是B的派生类   通过基类B的函数fun仍然可以访问
29. }

1.2.  **Base的友元可以通过Base的派生类Drived访问Base的private，protect成员变量，但不能访问Drived的private，protect成员变量。（这一点似乎与《C++ primer》里说的有点冲突）**

个人理解：Drived的对象本身就包含Base，Base的友元Frnd自然就可以访问Base的部分。

**[cpp]**  [***view plain***](http://www.voidcn.com/article/p-ktcxihws-bgr.html)  [***copy***](http://www.voidcn.com/article/p-ktcxihws-bgr.html)

1. #include <iostream>
2. **using** **namespace** std;
3. **class** Base
4. {
5. **int** m\_a;
6. **public**:
7. Base(**int** x=0){ m\_a=x; }
8. **friend** **class** Frnd;
9. };
11. **class** Drived:**public** Base
12. {
13. **private**:
14. **int** m\_c;
15. **public**:
16. Drived(**int** x):Base(x){m\_c=x;}
17. };
19. **class** Frnd
20. {
21. **public**:
22. **void** fun(Base& ob) { cout <<ob.m\_a << endl; }
23. **void** fun2(Drived& ob)
24. {
25. cout << ob.m\_a<<endl;
26. //cout <<ob.m\_c<<endl; //编译错误
27. }
28. };
30. **int** main()
31. {
32. Drived d(1);
33. Frnd f;
34. f.fun(d);
35. f.fun2(d);
36. system("pause");
37. **return** 0;
38. }

3 友元类的传递问题

A的友元是B，B的友元是C，那A的友元是C？　不是，友元类不具有传递性。