**一、友元的优缺点**

采用类的机制后实现了数据的隐藏与封装，类的数据成员一般定义为私有成员，成员函数一般定义为公有的，依此提供类与外界间的通信接口。有时候，**其他**类或者函数要访问该类的**非公有成员**，可以令其他类或者函数成为它的**友元（friend）**来实现。这样做提高了程序的运行效率，但同时也坏了类的**封装性和隐藏性**，使得非成员函数或其他类可以访问类的私有成员。

**二、友元函数**

友元定义格式如下：

friend 类型 函数名(形参);

首先，友元函数可以访问类的私有成员，其本身是定义在类外的普通函数，但是友元声明只能出现在类定义的内部，同时因为，友元不是类的成员也不受它所在区域访问控制级别的约束，其出现的位置不定，即既可以出现在private下也可以在public下；其次，若类想把一个函数作为它的友元，只需增加一条以friend关键字开头的函数声明语句即可。如：

class A

{

public:

friend void set\_show(int x, A &a); //该函数是友元函数的声明

private:

int data;

};

void set\_show(int x, A &a) //友元函数定义，为了访问类A中的成员

{

a.data = x;

cout << a.data << endl;

}

int main(void)

{

A a;

set\_show(1, a);

system("PAUSE");

return 0;

}

 这里值得注意的几点是：

（1）友元的声明仅仅指定了访问的权限，而非一个通常意义上的函数声明，如果我们希望类的用户能够调用某个友元函数，那么我们就必须在友元声明之外再专门针对函数进行一次声明。为了使友元对类的用户可见，我们通常把友元的声明与类的本身放置在同一个头文件中（类的外部）。

（2）友元函数的调用与一般函数的调用方式和原理一致。

（3）类和非成员函数的声明不是必须在它们的友元声明之前。

（4）友元函数能定义在类的内部，这样的函数是隐式内联的。

**三、友元类**

1、类之间的友元关系

如果希望一个类可以访问另一个类的非公有成员在内的所有成员（主要是非公有的成员），可以将一个类指定为另一类的友元类。

如将类B定义为类A的友元类：

class A

{

public:

friend class C; //这是友元类的声明

private:

int data;

};

class C //友元类定义，为了访问类A中的成员

{

public:

void set\_show(int x, A &a);

};

void C::set\_show(int x, A &a)

{

a.data = x;

cout << a.data << endl;

}

int main(void)

{

A a;

C c;

c.set\_show(1, a);

system("PAUSE");

return 0;

}

 这里使用友元类有几点要注意的：

（1）友元关系不存在传递性，即每个类负责控制自己的**友元类或者友元函数**；

（2）友元关系是单向的，不具有交换性。若类B是类A的友元，类A不一定是类B的友元，要看在类中是否有相应的声明。

（3）友元关系不能被继承。

2、成员函数作为友元

即一个类的成员函数作为另一个类的友元。此时，我们必须指出该成员函数属于哪个类。

当用到友元成员函数时，需注意友元声明和友元定义之间的相互依赖，下面的例子中，类B必须先定义，否则类A就不能将一个B的函数指定为友元。然而，只有在定义了类A之后，才能定义类B的该成员函数。更一般的讲，必须先定义包含成员函数的类，才能将成员函数设为友元。另一方面，不必预先声明类和非成员函数来将它们设为友元。

举例：

class A; //当用到友元成员函数时，需注意友元声明与友元定义之间的互相依赖。这是类A的声明

class B

{

public:

void set\_show(int x, A &a); //该函数是类A的友元函数

};

class A

{

public:

friend void B::set\_show(int x, A &a); //该函数是友元成员函数的声明

private:

int data;

void show();

};

void A::show()

{

cout << data << endl;

}

void B::set\_show(int x, A &a) //只有在定义类A后才能定义该函数，毕竟，它被设为友元是为了访问类A的成员

{

a.data = x;

cout << a.data << endl;

}

int main(void)

{

class A a;

class B b;

b.set\_show(1, a);

system("PAUSE");

return 0;

}