Access Control and Inheritance

Just as each class controls the initialization of its own members each class also controls whether its members are **accessible** to a derived class. (很显然, 每个类都要对其成员具有控制权)

这里, 首先给出继承权限的规则:

- `public`继承不会改变基类成员的访问属性
- `protected`继承将基类中的`public`成员变为派生类的`protected`成员, 其他成员权限不变
- `private`继承将基类中所有成员变为派生类中的`private`成员

`protected` Members

A class uses `protected` for those members that it is willing to share with its derived classes but wants to protect from general access. The `protected` specifier can be viewd as a blend of `private` and `public`.

- Like `private`, `protected` members are inaccessible to users of the class

```
class Foo{
    private:
        int val;// 对于类内成员可访问，或友元
    protected:
        string s;// 对于类内成员可访问，或友元
};
Foo f;
f.val;// error, private members are inaccessible to users
f.s;// error,....
```

- Like `public`, `protected` members are accessible to members and friends of classes derived from this class

In addition, `protected` has another important property:

- A derived class member or friend may access the `protected` members of the base class only through a derived object. The derived class has no special access to the `protected` members of base-class objects.

```
#include <iostream>
class Base {
private:
    int priv_mem;
protected:
    int prot_mem;
public:
    int pub_mem;
    Base(const int &v=0):priv_mem(v),prot_mem(v),pub_mem(v){}
};

class Derived :private Base {
```

```
    friend void fun(Derived&);
    friend void func(Base&);
public:
    Derived(const int &v=0):Base(v){}

};

void fun(Derived& obj) {
    // Derived 友元可以通过派生类对象访问基类的public和protected成员
    // 但是不可以访问private成员
    //std::count << "private mem:" << obj.priv_mem << std::endl;// error
    std::cout << "public mem:"<<obj.pub_mem << std::endl;
    std::cout << "protected mem:" << obj.prot_mem << std::endl;
}

void func(Base& obj) {
    // 当derived-to-base conversion可用时，即实参只能为derived类型时，
    // 派生类友元可以通过基类对象访问访问派生类对象中的base-class部分中的public成员
    // 当转换不可用时，实参只能为Base类型，此时无所谓是否为Derived的友元函数，都可以
    // 使用基类对象访问其public成员，但是，不可以访问base-class部分的protected成员

    // 因此，本质上就是派生类友元不可以通过基类对象访问该对象的protected成员
    //std::cout << obj.prot_mem << std::endl;// error
    std::cout << obj.pub_mem << std::endl;// ok, public member can be accessed
}
int main() {
    Derived d;
    Base b;
    func(b);
    fun(d);
}
```

基类的私有成员只可以被基类成员访问，其派生类和派生类的友元都不可以访问。

首先，派生类成员可以访问基类的`public`和`protected`成员，而`void fun(Derived &)`是派生类的友元，不可以访问基类的`private`。可以认为，派生类可以决定这些派生类对象的base-class部分的访问权限。

可以假设一下，如果派生类及其友元可以访问基类对象的`protected`成员，那么我们就可以通过该派生类及其友元改变基类的成员，也就是说，突破基类的访问保护，这显然是不合理的。

因此，C++规定，派生类及其友元只可以访问派生类对象中的base-class部分的受保护成员；对于普通的基类对象中的成员不具有特殊的访问权限。

public, private, and protected Inheritance

Access to a member that a class inherits is controlled by

- the access specifier for that member in the base class
- the access specifier in the derivation list of the derived class

```
class Base{
    public:
```

```
        void pub_mem();// public member
    protected:
        int prot_mem;// protected member
    private:
        char priv_mem;// private member
};

struct Pub_Derv: public Base{

    // all members are public by default.

    // ok, derived classes can access protected members
    int f(){return prot_mem;}

    // error, private members are inaccessible to derived classes
    char g(){return priv_mem;}
};

struct Priv_Derv: private Base{
    // all members are public by default

    // ok, private derivation doesn't affect access in the derivate class
    int f1()const {return prot_mem;}
};
```

The derivation access specifier has no effect on whether members (and friends) of a derived class may access the members of its own direct base class. Access to the members of a base class is controlled by the access specifiers in the base class itself. Both `Pub_Derv` and `Priv_Derv` may access the `protected` member `prot_mem`. Neither may access the `private` member `priv_mem`.

The purpose of the derivation access specifier is to control the access that users of the derived class-including other classes derived from the derived class-have (to the members inheritted from `Base`):

```
Pub_Derv d1; // members inheritted from Base are public
Priv_Derv d2;// members inheritted from Base are private
d1.pub_mem();// ok, pub_mem is public in the derived class
d2.pub_mem();// error, pub_mem is private in the derived class
```

Both `Pub_Derv` and `Priv_Derv` inherit the `pub_mem` function. When the inheritance is `public`, members retain their access specification.

The derivation access specifier used by a derived class also controls access from classes that inherit from that derived class: (派生访问符还可以控制继承该派生类的类的访问权限)

```
struct Derived_from_Public:public Pub_Derv{
    // ok, Base::prot_mem remains protected in Pub_Derv
    int use_base(){return prot_mem;}

};
```

```
struct Derived_from_private:public Priv_Derv{
    // error, Base::prot_mem is private in Priv_Derv
};
```

Classes derived from `Pub_Derv` may access `prot_mem` from `Base` because that member remains a `protected` member in `Pub_Derv`. In contrast, classes derived from `Priv_Derv` have no such access. To them, all the members that `Priv_Derv` inheritted from `Base` are `private`.

Had we defined another class, say, `Prot_Derv`, that used `protected` inheritance, the `public` members of `Base` would be `protected` members in that class. Users of `Prot_Derv` would have no access to `pub_mem`, but the members and friends of `Prot_Derv` could access that inherited member. (对于一个类而言，类的实例，或称为用户，不可以直接访问其`private`成员和`protected`成员)

Accessibility of Derived-to-Base Conversion

Whether the derived-to-base conversion is accessible depends on which code is trying to use the conversion and may depend on the access specifier used in the derived class' derivation. Assuming `D` inherits from `B`:

- user code may use the derived-to-base conversion only if `D` inherits publicly from `B`. User code may not use the conversion if `D` inherits from `B` using either `protected` or `private`.
- member functions and friends of `D` can use the conversion to `B` regardless of how `D` inherits from `B`. The derived-to-base conversion to a direct base class is always accessible to members and friends of a derived class.
- member functions and friends of classes derived from `D` may use the derived-to-base conversion if `D` inherits from `B` using either `public` or `protected`. Such code may use the conversion if `D` inherits privately from `B`.

Friendship and Inheritance

Just as friendship is not transitive, friendship is also not inherited. Friends of the base have no special access to members of its derived classes, and friends of a derived of a derived class have no special access to the base class:

```
class Base{
    // other members as before
    friend class Pal;// Pal has no access to classes derived from Base
};

class Pal{
    public:
        inf f(Base b){return b.prot_mem;}// ok, Pal is a friend of Base;
        int f2(Sneaky s){return s.j;}// error, Pal is not a friend of Sneaky

        // access to a base class is controlled by the base class, even inside a
derived object
        int f3(Sneaky s){return s.prot_mem;}// ok, Pal is a friend of Base
};
```

The fact that `f3` is legal may seeming surprising, but it follows directly from the notion that each class controls access to its own members. `Pal` is a friend of `Base`, so `Pal` can access the members of `Base` objects. That access includes access to `Base` objects that are embedded in an object of a type derived from `Base`.

Note:

> 派生类及其友元不能通过基类对象，修改基类对象的值，即不能访问基类成员。而基类的友元可以通过派生类对象修改该对象的基类部分的成员。

```
class Base {
private:
    int priv_mem;
protected:
    int prot_mem;
    friend class Pal;
public:
    Base(const int &v=0):priv_mem(v),prot_mem(v){}
};
class Derived :public Base {
    friend void clobber(Derived &);
    //friend void clobber_(Base &);//派生类及其友元不可以通过Base对象访问Derived成员

    int j;
public:
    Derived(const int &v=0):Base(v),j(v){}
};

void clobber(Derived& b) {
    b.j = b.prot_mem = 0;
}


class Pal {
public:
    int f(Base b) { return b.prot_mem; }

    //  int f2(Derived s) { return s.j; }

    int f3(Derived s) {
        return s.priv_mem;
    }// Pal是Base的友元，可以
    // 通过派生类对象访问派生类中基类部分的全部成员，但是不可以访问派生类中的派生类定义的成员

    int f4(Derived s){
        return s.j;// error
    }
};
```

Exempting Individual Members

(???)改变个别成员的可访问性?

Sometimes we need to change the access level of a name that a derived class inherits. We can do so by providing a `using` declaration.

```
class Base{
    public:
        std::size_t size()cosnt {return n;}
    protected:
        std::size_t n;
};

class Derived:Private Base{
    // private inheritance
    public:
        // maintain access levels for members related to the size of the object
        using Base::size;
    protected:
        using Base::n;
};
```

Because `Derived` uses `private` inheritance, the inherited members, `size` and `n`, are (by default) `private` members of `Derived` (默认情况下，所有成员都将被继承为`private`成员). The `using` declaratinons adjust the accessibility of these members. Users of `Derived` can access the `size` member, and classes subsequently derived from `Derived` can access `n`.

A `using` declaration inside a class can name any accessible (e.g.,not `private`) member of a direct or indirect base class.

> A derived class may provide a `using` declaration only for names it is permitted to access.
> 这是因为，基类要保证其类内`private`成员的访问权限。如果可以使用`using`声明更改其访问属性。则明显违背各类控制各类中的成员的原则。而基类中的`public`成员本身就可以被类的用户、类的派生类、类的友元、类的成员等访问。基类中的`protected`成员可以被类的成员、类的友元、类的派生类访问，但不可被类的用户访问。因此，此时派生类的`using`声明可以更改其访问属性。更改规则如下：

- 声明在`public`区域的，可以被该派生类的成员、用户、友元等等访问。
- 声明在`private`区域的，只可以被该派生类的成员、友元访问。
- 声明在`protected`区域的，可以被该派生类的成员、友元、其派生类访问。

其实这个就是将基类的`public`和`protected`成员的访问属性进行更改。此外，继承声明中会更改基类中的访问属性。

```
#include <iostream>
class Base {
private:
    int priv_mem;
protected:
    int prot_mem;
public:
    int pub_mem;

    Base(const int &v=0):priv_mem(v),prot_mem(v),pub_mem(v){}
```

```
};

class Derived :private Base {
public:
    Derived(const int &v=0):Base(v){}
    using Base::prot_mem;
    using Base::pub_mem;
protected:
    //using Base::pub_mem;

};
int main() {
    Derived d;
    std::cout << d.prot_mem << std::endl;
}
```

可以看到,using声明甚至可以将protected属性的成员更改为public属性。

Default Inheritance Protection Levels

By default, a derived class defined with the `class` keyword has `private` inheritance; a derived class defined with `struct` has `public` inheritance:

```
class Base{/*...*/};
struct D1:Base{/*...*/};// public inheritance by default
class D2:Base{/*...*/};// private inheritance by default
```

> class和struct关键字定义类,只有默认成员访问权限的区别。继承访问权限类似。