【C++】Day74

Class	C++
Date	@March 15, 2022
Material	
Series Number	
Summary	Access to members and derived-to-base conversion

【Ch15】OOP

15.5 Access Control and Inheritance

Each class controls whether its members are accessible to a derived class.

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 thought of as as blend of `private` and `public`:

- Like `private`, `protected` members are inaccessible to users of the class.
- Like `public`, `protected` members are accessible to members and friends of class 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 objects. The derived class has no special access to the protected members of base-class objects.

To understand this last rule, consider the following example:

```
class Base {
protected:
  int prot_mem; // protected member
};

class Sneaky : public Base {
  friend void clobber(Sneaky &); // can access Sneaky::prot_mem
  friend void clobber(Base&); // cannot access Base::prot_mem
  int j; // j is private by default
};
```

Members and friends of a derived class can access the protected members only in base-class objects that are embedded inside a derived type object.

public, private, and protected Inheritance

Access to a member that a class inherits is controlled by a combination of the access specifier for that member in the base class, and the access specifier in the derivation list of the derived class.

Consider the following hierarchy:

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

The derivation access specifier has no effect on whether members 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.

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 inherited from `Base`:

```
Pub_Derv d1; // members inherited from Base are public
Priv_Derv d2; // members inherited 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.
```

The derivation access specifier used by a derived class also controls access from classes that inherit from that derived class:

```
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_Dev
  int use_base() { return prot_mem; }
};
```

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` inherited from `Base` are `private`. To them, all the members that `Priv_Derv` inherited from Base are `private`.

If we define 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.

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 not use the conversion if `D` inherits privately from `B`.

Tip: For any given point in the code, if a public member of the base class would be accessible, then the derived-to-base conversion is also accessible.