【C++】Day74(2)

Class	C++
Date	@March 15, 2022
Material	
Series Number	
Summary	Friendship in Inheritance, Exempt Individual Members, and Default Protection

【Ch15】OOP

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 class have no special access to the base class.

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

class Pal {
public:
  int f(Base b) { return b.prot_mem; } // ok: Pal is a friend of Base
  int f2(Sneaky s) { return s.j; } // error: Pal not friend of Sneaky
  int f3(Sneaky s) { return s.prot_mem; } // ok: Pal is a friend
};
```

The fact that `f3` is legal may seem 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`.

When a class makes another class a friend, it is only that class to which friendship is granted. The base classes of, and classes derived from, the friend have no special access to the befriending class:

```
// D2 has no access to protected or private members in Base
class D2 : public Pal {
public:
  int mem(Base b) { return b.prot_mem; } // error: friendship doesn't inherit
};
```

Note: Friendship is not inherited; each class controls access to its members.

Exempting Individual Members

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 {
public:
  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.` The `using` declarations adjust the accessibility of these members. Users of `Derived` can access the `size` member, and classes subsequently derived from `Derived` can access `n`.

Note: A derived class may provide a using declaration only for names it is permitted to access.

Default Inheritance Protection Levels

The default derivation specifier depends on which keyword is used to define a derived class.

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
```

The only difference between using the `struct` and `class` keyword are the default access specifiers for members and the default derivation access specifier. There are no other distinctions.

Best Practices: A privately derived class should specify private explicitly rather than rely on the default. Being explicit makes it clear that private inheritance is intended and not an oversight.