【C++】Day25(2)

Class	C++
Date	@December 16, 2021
Material	
Series Number	
Summary	

【Ch7】Classes

7.4 Class Scope

Every class defines its own new scope. Outside the class scope, ordinary data and function members may be accessed only through an object, a reference, or a pointer using a member access operator.

We access type members from the class using the scope operator

```
Screen::pos ht = 24, wd = 80; //use the pos type defined by Screen
Screen(scrht, wd, ' ');
```

Scope and Members Defined outside the Class

The fact that a class is a scope explains why we must provide the class name as well as the function name when we define a member function outside its class. Outside of the class, the names of the members are hidden.

Recall the clear member of class Window_mgr. That function's parameter uses a type that is defined by `Window_mgr`:

```
void Window_mgr::clear(ScreenIndex i) {
  Screen &s = screens[i];
```

```
  s.contents = string(s.height * s.width, ' ');
}
```

Because the compiler sees the parameter list after noting that we are in the scope of the Class `Window_Mgr`, there is no need to specify that we want the `ScreenIndex` that is defined by `Window_Mgr`.

On the other hand, the return type of a function normally appears before the function's name. When a member function is defined outside the class body, any name used in the return type is outside the class scope. As a result, the return type msut specify the class of which it is a member.

For example, we might give `Window_mgr` a function, named `addScreen`, to add another screen to the display.

```
class Window_mgr {
public:
  //add a Screen to the window and returns its index
  ScreenIndex addScreen(const Screen&);
};

//return type is seen before we're in the scope of Window_mgr
Window_mgr::ScreenIndex Window_mgr::addScreen(const Screen& s) {
  screens.push_back(s);
  return screens.size() - 1;
}
```

7.4.1 Name Lookup and Class Scope

Note: Member function definitions are processed after the compiler processes all of the declarations in the class.

Because member function bodies are not processed until the entire class is seen, they can use any name defined inside the class.

If a member declaration uses a name that has not yet been seen inside the class, the compiler will look for that name in the scope in which the class is defined.

Name Lookup for Class Member Declarations

Names used in declarations, including name used for the return type and types in the parameter list, must be seen before they are used. If a member declarant uses a name that has not yet been seen inside the class, the compiler will look for that name in the scope in which the class is defined.

Type Names Are Special

Ordinarily, an inner scope can redefine a name from an outer scope even if that name has already been used in the inner scope.

However, in a class, if a member uses a name from an outer scope and that name is a type, then the class may not subsequently redefine that name:

```
typedef double Money;
class Account {
public:
  Money balance() { return bal; } //uses Money from the outer scope
private:
  typedef double Money; //error: cannot redefine Money
  Money bal;
}
```

Note: Even though this is an error, some compilers may quietly accept this code, even though the program is in error.

Normal Block-Scope Name Lookup inside Member Definitions

A name used in the body of a member function is resolved as follows:

1. First, look for a declaration of the name inside the member function. As usual, only declarations in the function body that precede the use of the name are considered
2. If the declaration is not found inside the member function, look for a declaration inside the class. All the members of the class are considered.
3. If a declaration for the name is not found in the class, look for a declaration that is in scope before the member function definition.

We can use this to override parameter with the same name in a member function:

```
void Screen::dummy_func(pos height) {
  cursor = width * this->height; //uses height defined in the Screen class
}
```

Note: Even though the class member is hidden, it is still possible to use that member by qualifying the member's name with the name of its class or by using the this pointer explicitly.

If we want to use the global height, we can ask for it explicitly using the scope operator:

```
void Screen::dummy_func(pos height) {
  cursor = width * ::height; //the global height
}
```

Note: Even though the outer object is hidden, it is still possible to access that object by using the scope operator.