【C++】Day76

Class	C++
Date	@March 18, 2022
Material	
Series Number	
Summary	Virtual Constructor and Synthesized copy, control

【Ch15】OOP

15.7 Constructors and Copy Control

15.7.1 Virtual Destructors

The base class generally should define a virtual destructor. The destructor needs to be `virtual` to allow objects in the inheritance hierarchy to be dynamically allocated.

If we call `delete` on a pointer to a dynamically allocated object, it is possible that the static type of the pointer might differ from the dynamic type of the object being destroyed.

For example, if we delete a pointer of type `Quote*`, that pointer might point at a `Bulk_quote` object.

If the pointer points at a `Bulk_quote`, the compiler has to know that it should run the `Bulk_quote` destructor.

```
class Quote {
public:
  // virtual destructor needed if a base pointer pointing to a derived object is deleted
  virtual ~Quote() = default;
};
```

So long as the destructor is `virtual`, when we delete a pointer to base, the correct destructor will be run:

```
Quote *itemP = new Quote; // same static and dynamic type
delete itemP; // destructor for Quote called
```

```
itemP = new Bulk_quote; // static and dynamic types differ
delete itemP; // destructor for Bulk_quote called
```

Warning: Executing delete on a pointer to base that points to a derived object has undefined behaviour if the base's destructor is not virtual.

15.7.2 Synthesized Copy Control and Inheritance

The synthesized members initialize, assign, or destroy the direct base part of an object by using the corresponding operation from the base class. For example,

The synthesized `Bulk_quote` default constructor runs the `Dis_Quote` default constructor, which in turn runs the `Quote` default constructor

Base Classes and Deleted Copy Control in the Derived

The way in which a base class is defined can cause a derived-class member to be defined as deleted:

- If the default constructor, copy constructor, copy-assignment operator, or destructor in the base class is deleted or inaccessible, then the corresponding member is defined as deleted. Because the compiler cannot use the base-class member to construct, assign, or destroy the base-class part of the object.
- If the base class has an inaccessible or deleted destructor, then the synthesized default and copy constructors in the derived classes are defined as deleted.

As an example,

```
class B {
public:
  B();
  B(const B&) = delete;
};

class D : public B {
  // no constructors
};

D d; // ok: D's synthesized default constructor uses B's default constructor
D d2(d); // error: D's synthesized copy constructor is deleted
D d3(std::move(d)); // error: implicitly uses D's deleted copy constructor
```