【C++】Day70

Class	C++
Date	@March 8, 2022
Material	
Series Number	
Summary	

【Ch15】OOP

15.2 Defining Base and Derived Classes

15.2.1 Defining a Base Class

We'll start by completing the definition of our `Quote` class:

```
class Quote {
public:
  Quote() = default;
  Quote(const std::string &book, double sales_price) : bookNo(book), price(sales_price) {}

  std::string isbn() const { return bookNo; }
  virtual double net_price(std::size_t n) const { return n * price; }
  virtual ~Quote() = default;
private:
  std::string bookNo;
protected:
  double price = 0.0;
};
```

We'll explain virtual destructor in a later chapter, but for now it is worth nothing that classes used as the root of an inheritance hierarchy almost always define a virtual destructor.

Member Functions and Inheritance

Derived classes inherit the members of their base class.

However, a derived class needs to be able to provide its own definition for operations, such as `net_price`, that are type dependent. In such cases, the derived class needs to override the definition it inherits from the base class, by providing its own definition.

The base class defines as virtual those functions it expects its derived classes to override.

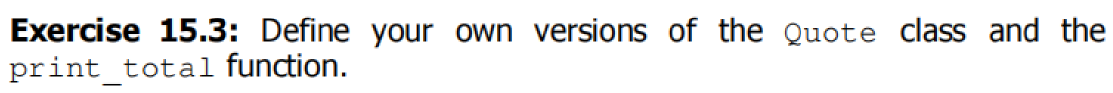
Any nonstatic member function other than a constructor may be virtual. The virtual keyword appear only on the declaration inside the class and may not be used on a function definition that appears outside the class body.

Access Control and Inheritance

Like any other code that uses the base class, a derived class may access the public members of its base class but may not access the private members.

However, sometimes a base class has members that it wants to let its derived classes use while prohibiting access to hose same members by other users. We specify such members after a protected access specifier.

Exercise


Exercise 15.3: Define your own versions of the `Quote` class and the `print_total` function.

See 15_3.cpp for code

15.2.2 Defining a Derived Class

A derived class must specify from which class(es) it inherits. It does so in its class derivation list.

Each base class name may be preceded by an optional access specifier, which is one of `public`, `protected`, or `private`.

A derived class must declare each inherited member function it intends to override. Therefore, our `Bulk_quote` class must include a `net_price` member.

```
class Bulk_quote : public Quote {
  Bulk_quote() = default;
  Bulk_quote(const std::string&, double, std::size_t, double);
  // overrides the base version in order to implement the bulk purchase discount policy
  double net_price(std::size_t) const override;
private:
  std::size_t min_qty = 0; // minimum purchase for the discount to apply
  double discount = 0.0;
};
```

Our `Bluk_quote` class inherits the `isbn` function and the `bookNo` and `price` data members of its `Quote` base class.

It defines its own version of `net_price` and has two additional data members, `min_qty` and `discount`.

The access specifier determines whether users of a derived class are allowed to know that the derived class inherits from its base class.

When the derivation is `public`, the `public` members of the base class become part of the interface of the derived class as well. In addition, we can bind an object of a publicly derived type to a pointer or reference to the base type.

Because we used `public` in the derivation list, the interface to `Bulk_quote` implicitly contains the `isbn` function, and we may use a `Bulk_quote` object where a pointer or reference to `Quote` is expected.

Virtual Functions in the Derived Class

Derived classes frequently, but not always, override the virtual functions that they inherit. If a derived class does not override a virtual from its base, then the derived class inherits the version defined in its base class.

Derived-Class Objects and the Derived-to-Base Conversion

A derived object contains multiple parts: a subobject containing the members defined in the derived class itself, plus subobjects corresponding to each base class from which the derived class inherits.

Thus, a `Bulk_quote` object will contain four data elements: the `bookNo` and price data members that it inherits from `Quote`, and the `min_Qty` and `discount` members, which are defined by `Bulk_quote`.

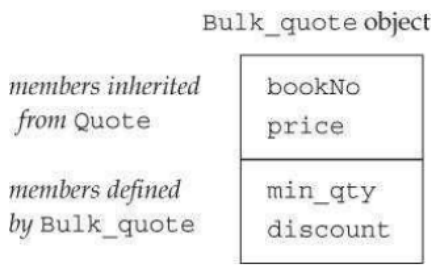


Figure 15.1. Conceptual Structure of a Bulk_quote Object

Because a derived object contains subparts corresponding to its base class, we can use an object of a derived type as if it were an object of its base type.

```
Quote item;
Bulk_quote bulk;
Quote *p = &bulk;
```

This conversion is often referred to as the derived-to-base conversion.

Note: The fact that a derived object contains subobjects for its base classes is key to how inheritance works.