【C++】Day69(3)

Class	C++
Date	@March 7, 2022
Material	
Series Number	
Summary	

【Ch15】Object-Oriented Programming

OOP is based on three fundamental concepts: data abstraction, inheritance, and dynamic binding.

15.1 OOP: An Overview

Inheritance

Classes related by inheritance form a hierarchy. Typically there is a base class at the root of the hierarchy, from which the other classes inherit, directly or indirectly.

These inheriting classes are known as derived classes. The base class defines those members that are common to the types in the hierarchy. Each derived class defines those members that are specific to the derived class itself.

The base class defines as `virtual` those functions it expects its derived classes to define for themselves.

```
class Quote {
public:
  std::string isbn() const;
  virtual double net_price(std::size_t n) const;
};
```

A derived class must specify the class from which it intends to inherit. It does so in a class derivation list, which is a colon followed by a comma-separated list of base

classes each of which may have an optional access specifier.

```
class Bulk_quote : public Quote {
public:
  double net_price(std::size_t) const override;
};
```

Because `Bulk_quote` uses `public` in its derivation list, we can use objects of type `Bulk_quote` as if they were `Quote` objects.

A derived class must include in its own class body a declaration of all the virtual functions it intends to define for itself.

Dynamic Binding

Through dynamic binding, we can use the same code to process objects of either type `Quote` or `Bulk_quote` interchangeably.

For example, the following function prints the total price for purchasing the five number of copies of a given book:

```
// calculate and print the price for the given number of copies, applying any discounts
double print_total(ostream &os, const Quote &item, size_t n) {
  // depending on the type of the object bound to the item parameter
  // calls either Quote::net_price or Bulk_quote::net_price
  double ret = item.net_price(n);
  os << "ISBN: " << item.isbn() << " # sold: " << n << " total due: " << ret << endl;
  return ret;
}
```

Because the item parameter is a reference to `Quote`, we can call this function on either a `Quote` object or a `Bulk_quote` object.

The version of `net_price` that is run will depend on the type of the object that we pass to `print_total`:

```
// basic has type Quote, bulk has type Bulk_quote
print_total(cout, basic, 20);
print_total(cout, bulk, 20); // calls Bulk_quote version of net_price
```

Because the decision as to which version to run depends on the type of the argument, that decision can’t be made until run time. Therefore, dynamic binding is sometimes known as run-time binding.

Note: In C++, dynamic binding happens when a virtual function is called through a reference to a base class.