【C++】Day23(2)

Class	C++
Date	@December 13, 2021
Material	
Series Number	
Summary	

【Ch7】Functions

7.2 Access Control and Encapsulation

At this point, we have defined an interface for our class; but nothing forces users to use that interface. Users can reach inside Sales_data object and meddle with its implementation.

In C++, we use access specifiers to enforce encapsulation:

- Members defined after a `public` specifier are accessible to all parts of the program. The public members define the interface to the class.
- Members defined after a `private` specifier are accessible to the member functions of the class but are not accessible to code that uses the class. The private sections encapsulate the implementation.

Redefine our Sales_data Class:

```
class Sales_data {
public: // access specifier added
  Sales_data() = default;
  Sales_data(const std::string &s, unsigned n, double p): bookNo(s), units_sold(n), revenue(p*n) { }
  Sales_data(const std::string &s): bookNo(s) { }
  Sales_data(std::istream&);
  std::string isbn() const { return bookNo; }
  Sales_data &combine(const Sales_data&);

private: // access specifier added
  double avg_price() const { return units_sold ? revenue/units_sold : 0; }
  std::string bookNo; unsigned units_sold = 0; double revenue = 0.0;
};
```

The constructors and member functions that are part of the interface follow the `public` specifier; the data members and the functions that are part of the implementation follow the `private`

specifier.

Using the class or struct Keyword

We used the `class` keyword rather than struct to open the class definition.

This change is strictly stylistic; we can define a class type using either keyword. The only difference between struct and class is the default access level.

A class may define members before the first access specifier. Access to such members depends on how the class is defined.

- If we use the `struct` keyword, the members defined before the first access specifier are public
- If we use the `class` keyword, then the members are private.

As a matter of programming style, when we define a class intending for all of its members to be public, we use struct.

If we intend to have private members, then we use class.

Note: The only difference between using class and using struct to define a class is the default access level.

7.2.1 Friends

A class can allow another class or function to access its nonpublic members by making that class or function a `friend`. A class makes a function its friend by including a declaration for that function preceded by the keyword friend:

```
class Sales_data {
  // friend declarations for nonmbmer Sales_data operations added
  friend Sales_data add(const Sales_data &, const Sales_data &);
  friend std::istream &read(std::istream&, Sales_data&);
  friend std::ostream &print(std::ostream&, const Sales_data&);

public: // access specifier added
  Sales_data() = default;
  Sales_data(const std::string &s, unsigned n, double p): bookNo(s), units_sold(n), revenue(p*n) { }
  Sales_data(const std::string &s): bookNo(s) { }
  Sales_data(std::istream&);
  std::string isbn() const { return bookNo; }
  Sales_data &combine(const Sales_data&);
```

```
private: // access specifier added
  double avg_price() const { return units_sold ? revenue/units_sold : 0; }
  std::string bookNo; unsigned units_sold = 0; double revenue = 0.0;
};
```