【C++】Day21

Class	C++
Date	@December 11, 2021
Material	
Series Number	
Summary	

【Ch7】Classes

Recap: Sales_data Class

```
struct Sales_data {
  //new members: operations on Sales_data objects
  std::string isbn() const { return bookNo; }
  Sales_data& combine(const Sales_data&);
  double avg_price() const;

  std::string bookNo;
  unsigned units_sold = 0;
  double revenue = 0.0;
};
```

Class Scope and Member Functions

Recall that a calss is iteself a scope. The definitions of the member functions of a class are nested inside the scope of the class itself.

Hence, `isbn`'s use of the name `bookNo` is resolved as the data member defined inside `Sales_data`.

Note that `isbn` can use `bookNo` even though bookNo is defined after isbn.

The compiler processes classes in two steps-the member declarations are compiled first, after which the member function bodies, if any, are processed.

Thus, member function bodies may use other members of their class regardless of where in the class those members appear.

Defining a Member Function outside the Class

When we define a member funcion outside the class body, the member's definition must match its declaration.

That is, the return type, parameter list, and name must match the declaration in the class body. If the member was declared as a const member function, then the definition must also specify const after the parameter list.

The name of a member defined outside the class must include the name of the class of which it is a member:

```
double Sales_data::avg_price() const {
  if(units_sold)
    return revenue / units_sold;
  else
    return 0;
}
```

The function name, `Sales_data::avg_price`, uses the scope operator `::` to say that we are defining the function named `avg_price` that is declared in the scope of the `Sales_data` class.

Defining a Function to Return "This" Object

The `combine` function is intended to act like the compound assignment operator, +=.

The object on which this function is called represents the left-hand operand of the assignment. The right-hand operand is passed as an explicit argument:

```
Sales_data& Sales_data::combine(const Sales_data &rhs) {
  units_sold += rhs.units_sold; //add the members of rhs into this
  revenue += rhs.revenue; //the members of "this" object
  return *this; //return the object on which the function was called
}
```

When our transaction-processing program calls:

```
total.combine(trans); //update the running total
```

the address of `total` is bound to the implicit `this` parameter and `rhs` is bound to trans.

We do not need to use the implicit this pointer to access the members of the object on which a member function is executing. However, we do need to use this to access the object as a whole:

```
return *this; //return the object on which the function was called.
```

Exercise

Exercise 7.4: Write a class named `Person` that represents the name and address of a person. Use a `string` to hold each of these elements. Subsequent exercises will incrementally add features to this class.

Exercise 7.5: Provide operations in your `Person` class to return the name and address. Should these functions be `const`? Explain your choice.

```
struct Person {
  string name;
  string address;

  const string name() const { return name; }
  string address() const { return address; }
};
```