【C++】Day64(2)

Class	C++
Date	@February 27, 2022
Material	
Series Number	
Summary	Arithmetic, Relational, and Assignment Operators

【Ch14】Overloaded Operations and Conversions

14.3 Arithmetic and Relational Operators

Ordinarily, we define the arithmetic and relational operators as nonmember functions in order to allow conversions for either the left- or right-hand operand.

These operators shouldn't need to change the state of either operand, so the parameters are ordinarily references to `const`.

An arithmetic operators usually generates a new value that is the result of a computation on its two operands. That value is distinct from either operand and is calculated in a local variable.

```
// assumes that both objects refer to the same book
Sales_data operator+(const Sales_data &lhs, const Sales_data &rhs) {
  Slaes_data sum = lhs; // copy data members from lhs into sum
  sum += rhs; // add rhs into sum
  return sum;
}
```

Tip: Classes that define both an arithmetic operator and the related compound assignment ordinarily ought to implement the arithmetic operator by using the compound assignment.

14.3.1 Equality Operators

Ordinarily, classes in C++ define the equality operator to test whether two objects are equivalent. They usually compare every data member and treat two objects as equal if and only if all the corresponding members are equal.

```
bool operator==(const Sales_data &lhs, const Sales_data &rhs) {
  return lhs.isbn() == rhs.isbn() && lhs.units_sold == rhs.units_sold && lhs.revenue == rhs.revenue;
}

bool operator!=(const Sales_data &lhs, const Sales_data &rhs) {
  return !(lhs == rhs);
}
```

Exercise

Exercise 14.16: Define equality and inequality operators for your `StrBlob` (§ 12.1.1, p. 456), `StrBlobPtr` (§ 12.1.6, p. 474), `StrVec` (§ 13.5, p. 526), and `String` (§ 13.5, p. 531) classes.

See 14_16.cpp for code

14.3.2 Relational Operators

Ordinarily the relational operators should

1. Define an ordering relation that is consistent with the requirements for use as a key to an associative container
2. Define a relation that is consistent with == if the class has both operators. In particular, if two objects are ≠, then one object should be < the other.

14.4 Assignment Operators

In addition to the copy- and move-assignment operators that assign one object of the class type to another object of the same type, a class can define additional assignment operators that allow other types as the right-hand operand.

As one example, the library vector class defines a third assignment operator that takes a braced list of elements. We can use this operator as follows:

```
vector<string> v;
v = { "a", "an", "the" };
```

We can add this operator to our `StrVec` class as well:

```
class StrVec {
public:
  StrVec &operator=(std::initializer_list<std::string>);
};
```

To be consistent with assignment for the built-in types, our new assignment operator will return a reference to its left-hand operand:

```
StrVec &StrVec::operator=(intializer_list<string> il) {
  auto data = alloc_n_copy(il.begin(), il.end());
  free();
  elements = data.first;
  first_free = cap = data.second;
  return *this;
}
```

Compound-Assignment Operators

Compound Assignment operators are not required to be members. However, we prefer to define all assignments, including compound assignments, in the class.

`Slaes_data` compound-assignment operator:

```
Sales_data &Sales_data::operator+=(const Sales_data &rhs) {
  units_sold += rhs.units_sold;
  revenue += rhs.revenue;
  return *this;
}
```

Best Practices: Assignment operators must, and ordinarily compound-assignment operators should, be defined as members. These operators should return a reference to the left-hand operand.