【C++】Day64

Class	C++
Date	@February 27, 2022
Material	
Series Number	
Summary	Overload the Input and Output Operators

【Ch14】Overloaded Operations and Conversions

14.2 Input and Output Operators

14.2.1 Overloading the Output Operator <<

Ordinarily, the first parameter of an output operator is a reference to a nonconst ostream object. The ostream is nonconst because writing to the stream changes its state.

The second parameter ordinarily should be a reference to `const` of the class type that we want to print.

To be consistent with other output operators, `operator<<` normally returns its `ostream` parameter.

The Sales_Data Output Operator

We'll write the Sales_data output operator:

```
ostream& operator<<(ostream &os, const Sales_data &item) {
  os << item.isbn() << " " << item.units_sold << "" << item.avg_price();
  return os;
}
```

Output Operators Usually Do Minimal Formatting

The output operators for the built-in types do little if any formatting. If the operator does print a newline, each user would be unable to print descriptive text along with the object in the same line.

Best Practice: Generally, output operators should print the contents of the object, with minimal formatting. They should not print a newline.

IO Operators Must Be Nonmember Functions

The input and output operators cannot be members of our own class. If they were, the left-hand operand would have to be an object of our class type:

```
Sales_data data;
data << cout;
```

Thus, we must define them as nonmember functions.

Exercise

Exercise 14.6: Define an output operator for your `Sales_data` class.

See 14_6.cpp for code

14.2.2 Overloading the Input Operator>>

The Sales_data Input Operator

```
istream &operator>>(istream &is, Sales_data &item) {
  double price;
  is >> item.bookNo >> item.units_sold >> price;
  if(is) // check that the inputs succeed
    item.revenue = item.units_sold * price;
  else
    item = Slaes_data(); // input failed: give the object the default state
  return is;
}
```

Note: Input operators must deal with the possibility that the input might fail; output operators generally don't bother.

Errors during Input

The kinds of errors that might happen in an input operator include the following:

- A read operation that might fail because the stream contains data of an incorrect type. For example, a character is provided when a numeric data is expected.
- Any of the reads could hit end-of-file or some other error on the input stream.

Rather than checking each read, we check once after reading all the data and before using those data:

```
if(is)
    item.revenue = item.units_sold * price;
else
  item = Sales_data();
```

Best Practices: Input operators should decide what, if anything, to do about error recovery.