[C++] Day63

• Class	C++
 □ Date	@February 25, 2022
Material	
# Series Number	
■ Summary	

[Ch14] Overloaded Operations and Conversions

Operator overloading lets us define the meaning of an operator when applied to operands of a class type.

14.1 Basic Concepts

Overloaded operators are functions with special names: the keyword operator followed by the symbol for the operator being defined. Like any other function, an overloaded operator has a return type, a parameter list, and a body.

An overloaded operator function has the same number of parameters as the operator has operands. A unary operator has one parameter; a binary operator has two. In a binary operator, the left-hand operand is passed to the first parameter and the right-hand operand to the second.

If an operator function is a member function, the first(left-hand) operand is bound to the implicit this pointer. Because the first operand is implicitly bound to this, a member operator function has one less(explicit) parameter than the operator has operands.

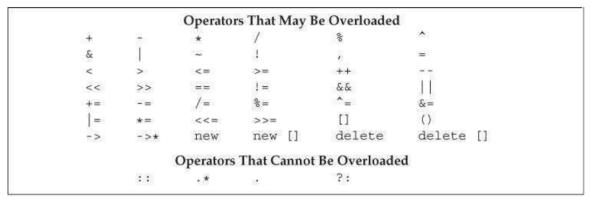
An operator function must either be a member of a class or have at least one parameter of class type.

```
// error: cannot redefine the built-in operator for ints
int operator+(int, int);
```

This restriction means that we cannot change the meaning of an operator when applied to operands of built-in type.

We can overload most, but not all, of the operators. The following table shows whether or not an operator may be overloaded.

Table 14.1. Operators



An overloaded operator has the same precedence and associativity as the corresponding built-in operator. Regarless of the operand types

```
x == y + z;
```

is always equal to x == (y + z);

Calling and Overloaded Operator Function Directly

Ordinarily, we "call" an overloaded operator function indirectly by using the operator on arguments of the appropriate type. However, we can also call an overloaded operator function directly in the same way that we call an ordinary function .

[C++] Day63

```
// equivalent calls to a nonmember opator function
data1 + data2;
operator+(data1, data2);
```

We call a member operator function explicitly in the same way that we call any other member function. We name an object(or pointer) on which to run the function and use the dot operator(or arrow) to fetch the function we wish to call.

```
data1 += data2;
data1.operator+=(data2); // equivalent call to a member operator function
```

Some Operators Shouldn't Be Overloaded

Recall that a few operators guarantee the order in which operands are evaluated. Because using an overloaded operator is really a function call, these guarantees do not apply to overloaded operators.

In particular, the operand-evaluation guarantees of the logical AND, logical OR, and comma operators are not preserved. Moreover, overloaded versions of a operators do not preserve short-circuit evaluation properties of the built-in operators. Both operands are always evaluated.

It is usually a bad idea to overload these operators.

Best Practices: Ordinarily, the comma, address-of, logical AND, and logical OR operators should not be overloaded.

Use Definitions That Are Consistent with the Built-in Meaning

- If the class does IO, define the shift operators to be consistent with how IO is done for the built-in types.
- If the class has an operation to test for equality, define operator==. If the class has operator==, it should usually have operator!= as well.

[C++] Day63

• If the class has a single, natural ordering operation, define operator<. If the class has operator<, it should probably have all of the relational operators.

[C++] Day63