南方科技大学
SOUTHERN UNIVERSITY OF SCIENCE AND TECHNOLOGY

C/C++ Program Design

Lab 10, operator overloading and friend function

廖琪梅，王大兴

Operator overloading and friend function

- Operator overloading
- Friend function
- Overloading << operator

Operator overloading

To overload an operator, use a special function form called an **operator function**.

return_type operator op(argument-list)

op is the symbol for the operator being overloaded

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

member function, non-member function, friend function

Only non-member operator overloading function can **implement type conversion** on its **left argument**, so if a function need convert type on its left argument, define the function as non-member function; if the function must get the non-public members of the class, define it as a friend function of the class.

Other cases beyond the above, define the function as a member function.

The assignment (=)operators must be defined as member function. However, IO operators(<< and >>) must be non-member functions.

```cpp
#pragma once

class Rational
{
private:
    int numerator;
    int denominator;

public:

    Rational(int numerator = 1, int denominator = 1)
    {
        this->numerator = numerator;
        this->denominator = denominator;

    }

    int getNumerator() const { return numerator; }
    int getDenominator() const { return denominator; }

    friend std::ostream& operator<< (std::ostream& os, const Rational& r);

    void Show() const;
};

Rational operator* (const Rational& lhs, const Rational& rhs);
```

Constructor with default arguments

Define << operator function as a friend function
Its declaration is inside the class and the definition is outside the class.

Define * operator function as a normal function


```
Rational operator* (const Rational& lhs, const Rational& rhs)
{
    int numer = lhs.getNumerator() * rhs.getNumerator();
    int deno = lhs.getDenominator() * rhs.getDenominator();
    return Rational(numer, deno);
}


std::ostream& operator<< (std::ostream& os, const Rational& r)
{
    os << r.numerator << "/" << r.denominator << std::endl;
    return os;

}
```

```
#include <iostream>
#include "rational.h"

using namespace std;

int main()
{
    //Rational class with operator overloading
    Rational oneHalf(1, 2);
    Rational oneThird(1, 3);

    Rational result1 = oneHalf * oneThird;
    Rational result2 = oneThird *  2;
    Rational result3 = 3 * oneHalf;

    cout << result1 << result2 << result3;

    return 0;
}
```

```
1/6
2/3
3/2
```

Note: In this example, the **Rational** class must have one argument constructor without explicit keyword. Otherwise, the int type can not be converted to the object type.

Memberwise initialization and objects assignment

If you use a existing object to initialize a new object, the compiler will invoke the copy constructor to implement the memberwise initialization. The data members of the class are copied in turn. This is called *default memberwise initialization*.

```
Rational other = oneHalf;
```

Invoke copy constructor implicitly

other.numerator = oneHalf.numerator;
other.denominator = oneHalf.denominator;

```
Rational(Rational& r)
{
    this->numerator = r.numerator;
    this->denominator = r.denominator;
}
```

The copy constructor of Rational class
If you don't provide the copy constructor, the compiler will provide a default one and implement the memberwise initialization.

This copy constructor is the same as the default copy constructor.
If the data member includes a pointer, you must provide the copy constructor in your class.

Memberwise initialization and objects assignment

If you use an equal sign to assign one object to another, this is called assign one object to another. In this case, the copy constructor is not invoked.

```
Rational same;
same = oneThird;
```

Assignment statement

same.numerator = oneThird.numerator;
same.denominator = oneThird.denominator;

Although this operation is also memberwise assignment, but it does not invoke the copy constructor. The assignment operator(=) can be overloaded. If the data member includes a pointer, you must overload the assignment operator in your class.

Exercise:

- Create a class called **Complex** for performing arithmetic with complex numbers. Write a program to test your class. Complex numbers have the form

  realPart + imaginaryPart * i

- Develop a complete class containing proper constructor functions as well as setter and getter functions. The class should also provide the following overloaded operator capabilities:

(1) Overload the addition operator (+) to add two Complex numbers.

(2) Overload the subtraction operator (-) to subtract two Complex numbers.

(3) Overload the assignment operator(=) to assign one Complex to another.

(4) Overload the multiplication operator (*) to multiply two Complex numbers.

Exercise:

(5)Overload the == and != operators to allow comparisons of Complex numbers.

(6) Modify the class to enable input and output of Complex numbers via overloaded **>>** and **<<** operators, respectively.

Write a test program to test your Complex class.