LAB 5

CLASS - 2

Department of Electronics Engineering
National Chiao Tung University

Why Operator Overloading?

- C++ can overload operators to perform different operations depending on their context and data types.
  - Overloadable operators:

+	-	*	/	%	^
&	|	~	!	,	=
<	>	<=	>=	++	--
<<	>>	==	!=	&&	||
+=	-=	/=	%=	^=	&=
|=	*=	<<=	>>=	[]	()
->	->*	new	new []	delete	delete []

Operator Overloading

- Define a member function operator+

```
class complex {
        double re, im;
public:
        complex(double r = 0.0, double i = 0.0) : re(r), im(i) { }
        const complex operator+(const complex&) const;
};
const complex complex::operator+(const complex& rhs) const {
        complex result(rhs); // using copy ctor, too
        result.re += re; result.im += im;
        return result;
 }
int main() {
        complex a(1, 1), b(2, 2), c;
        c = a.operator+(b); // ok! explicit call, just ugly!
        c = a + b; // ok! it is just a shorthand for operator+
}
```

Another Way for Operator Overloading

- Overloaded operators are NOT necessarily member functions!

```
class complex {
        double re, im;
public:
        complex(double r = 0.0, double i = 0.0) : re(r), im(i) { }
        double real() const { return re; }
        double image() const { return im; }
};
const complex operator+(const complex& lhs, const complex& rhs) {
        double real, image;
        real = lhs.real() + rhs.real(); image = lhs.image() + rhs.image();
        return complex(real, image);
}
int main() {
        complex a(1, 1), b(2, 2), c;
        c = operator+(a, b); // ok! explicit call, just ugly!
        c = a + b; // ok! it is just a shorthand for operator+
}
```

Returning Constant Value

- **const complex operator+(const complex& lhs , const complex& rhs)**
- **complex operator+(const complex& lhs , const complex& rhs)**

```
int main() {
        complex a(1,1), b(2,2), c(3,3);
        (a + b) = c; // no error if using red one; error if using blue one
        if((a+b) = c) // Oops, programmer actually wants => if((a+b) ==c)
        do_things // again, no error if using red one; error if using blue one
}
```

- Hence, blue one is preferred

Member vs. Nonmember Operators

- If mixed-mode arithmetic is allowed e.g., allow adding a complex with a double

```
int main() { // operator+ is a member function here
        complex a(1,1), b;
        b = a + 1.0; // ok! a.operator+( complex(1.0) )
        b = 1.0 + a; // error! 1.0.operator+(a) <= no such function!
}
int main() { // operator+ is a nonmember function here
        complex a(1,1), b;
        b = a + 1.0; // ok! operator+( a, complex(1.0) )
        b = 1.0 + a; // ok! operator+( complex(1.0), a )
}
```

- In general, nonmember version is preferred

Friend Functions (1/3)

- Nonmember functions

  - access private members through accessors and mutators

  - inefficient (overhead of calls to accessors and mutators)

- Friend functions can directly access private members

  - same access privilege as member functions

  - no calls to accessors and mutators =>more efficient

- You can make specific nonmember functions friends for better efficiency!

Friend Functions (2/3)

```
class complex {
    double re, im;
public:
    complex(double r = 0.0, double i = 0.0) : re(r), im(i) { }
    double real() const { return re; }
    double image() const { return im; }
    friend const complex operator+(const complex&, const complex&);
};
const complex operator-(const complex&, const complex&);
```

Friend Functions (3/3)

```
// no need to add friend prefix in function definition
const complex operator+(const complex& lhs, const complex& rhs) {
    complex result(lhs);
    result.re += rhs.re; result.im += rhs.im;
    return result;
} // a friend function has same access privilege as member functions

const complex operator-(const complex& lhs, const complex& rhs)
    double real = lhs.real() + rhs.real();
    double image = lhs.image() + rhs.image();
    return complex(real, image);
} // need accessors to get private data
```

Overload <<

```
std::ostream& operator<<(std::ostream& os, const complex& rhs) {
    os << rhs.real() << '+' << rhs.image() << 'i' ;
    return os;
}

int main(){
    complex a(2,3), b(4,5);
    cout << a << endl << b << endl; // more elegant!
}
```

- Output:

  2+3i

  4+5i

- It is common to make operator<< a friend

Return Value of Operator <<

- If you make operator<< return void …

```
void operator<<(ostream& os, const complex& rhs) {
        os << rhs.real() << '+' << rhs.image() << 'i' ;
}
int main() {
        complex a(2,3), b(4,5);
        cout << a << endl << b << endl; // compilation error!
}         void
```

Overload >>

- You can use “cin >>” for user-defined types
  - first, make istream& operator>>(istream&, complex&) a friend

```
istream& operator>>(istream& is, complex& rhs) {
        is >> rhs.re >> rhs.im ;
        return is;
}
int main() {
        complex a, b;
        cin >> a >> b;
        cout << a << endl << b << endl;
}
```

```
[M106ylu@eng02 Lab4]$ ./test2
1 2 3 4
1+2i
3+4i
```

Exercise (1/3)

- implement a class Complex and provide the following functions.
  - 2 private data members :
    - double re : real part
    - double im : Imaginary part
  - You have to implement functions below in operator.cpp
    - constructor Complex(double r, double i)
      - Take r as the real part and i as the Imaginary part
    - operator+, operator-, operator*, operator==, operator!= on Complex
    - operator! return the conjugate of the Complex
    - operator<< and operator>> for output/input Complex
  - You have to use makefile to compile your code, here is a makefile example for your reference.

```
CC = g++
CFLAGS = -g -Wall -O3
OBJS = operator.o
BINS = Lab05
all: $(BINS)

%.o: %.cpp %.h
    $(CC) $(CFLAGS) -c $< -o $@

$(BINS): main.cpp $(OBJS)
    $(CC) $(CFLAGS) $^ -o $@

run:
    ./$(BINS)
clean:
    rm $(BINS) $(OBJS)
```

Exercise (2/3)

- You should use friend functions and member functions when implementing

  operator **+**, **-**, *****, **==**, **!=**, **!**, **<<**, **>>**

- An example is given below

```
#include <iostream>
#include <fstream>

using namespace std;

#ifndef _OPERATOR_H_
#define _OPERATOR_H_
class Complex
{
public:
    Complex(){};
    Complex(double r, double i);
    friend const Complex operator-(const Complex &lhs, const Complex &rhs);
    friend const Complex operator!(const Complex &c);
    friend bool operator==(const Complex &lhs, const Complex &rhs);
    friend istream &operator>>(istream &in, Complex &rhs);
    friend ostream &operator<<(ostream &out, const Complex &rhs);
    const Complex operator+(const Complex &rhs);
    const Complex operator*(const Complex &rhs);
    bool operator!=(const Complex &rhs);

private:
    double re;
    double im;
};
#endif
```

Exercise (3/3)

- Main function is provided and don’t modify it.

```cpp
#include <iostream>
#include <fstream>
#include <string>
#include "operator.h"

int main()
{
    Complex a, b;
    cout << "Please cin Complex a and Complex b: "<< endl;
    cin >> a >> b;
    cout << "a = " << a << endl;
    cout << "b = " << b << endl;
    cout << "a + b = " << a + b << endl;
    cout << "a - b = " << a - b << endl;
    cout << "a * b = " << a * b << endl;
    cout << "conjugate of a = " << !a << endl;

    if (a == b)
        cout << "a is equal to b." << endl;
    if (a != b)
        cout << "a is not equal to b." << endl;

    return 0;
}
```

- Sample input/output

```
Please cin Complex a and Complex b:
1 2 3 4
a = 1+2i
b = 3+4i
a + b = 4+6i
a - b = -2-2i
a * b = -5+10i
conjugate of a = 1-2i
a is not equal to b.
```

Compile & Run & Demo

- Compile
  - g++ -std=c++11 main.cpp operator.cpp -I . -o Lab05
  - or using makefile
- Run
  - ./Lab05
- Demo (注意!! 指令有改)
  - /home/share/demo_OOP112_2 Lab 05
  - 以後的 lab/hw 需要自行編譯後產生執行檔，才使用OJ指令

```
[s410510026@mseda03 Lab05]$ g++ -std=c++11 main.cpp operator.cpp -I . -o Lab05
[s410510026@mseda03 Lab05]$ ls
Lab05  main.cpp  makefile  operator.cpp  operator.h  operator.o
[s410510026@mseda03 Lab05]$ /home/share/demo_OOP112_2 Lab 05

 Please compile your code first and generate the exe. file
 The exe. file must name as "Lab05"
 Test case must use "cin" for input
 Test case must use "cout" for output

===== Case 1 =====
 PASS
```

前一個版本/home/share/demo_OOP112 (只需 cpp) 支援到 Lab01-04, Hw01-05