Programming Methodology
Practice Session #12

Operator Overloading

Operator Overloading (1)

- Class를 피연산자로 가지는 특정 연산자의 기능을 재정의 함으로써 연산자를 통한 class의 사용을 가능하게 해 주는 기능.

- Operator overloading을 사용하면 보다 더 직관적인 코드를 작성할 수 있다.
  - ex) `Complex` class

```
c3 = c1.add(c2);
c4 = c1.sub(c2);
c5 = c1.mul(c2);
c6 = c1.div(c2);
```

Operator Overloading

```
c3 = c1 + c2;
c3 = c1 - c2;
c3 = c1 * c2;
c3 = c1 / c2;
```

Binary Operator Overloading (1)

- 두 개의 피연산자를 가지는 연산자를 **binary operator** 라 한다.
  - `+` , `-` , `*` , `/` 등

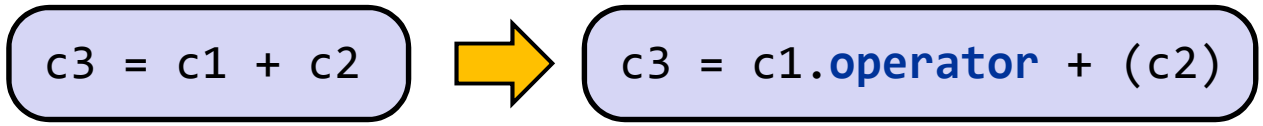
- 두 가지 방법으로 operator overloading을 구현 할 수 있다.
  - Member function을 이용하는 방법
  - Global function과 **`friend`** 키워드를 이용하는 방법

Binary Operator Overloading (2)

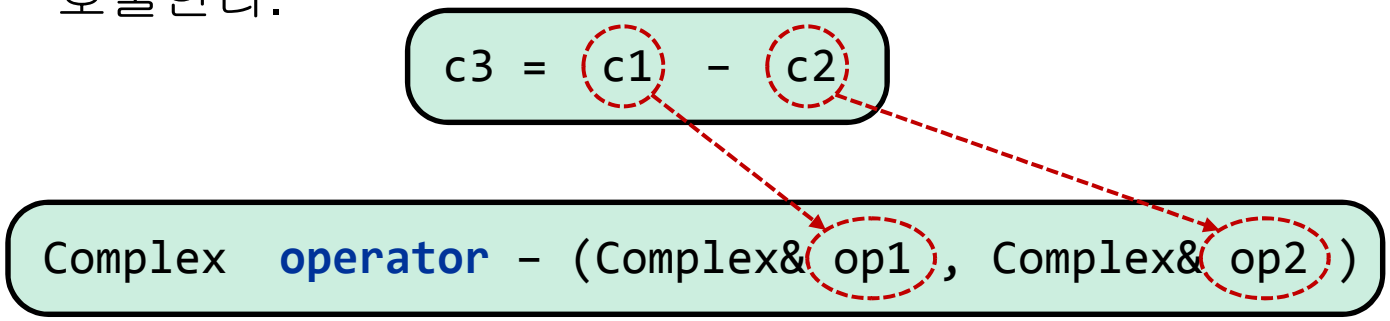
```
class Complex
{
    private:
        double real, image;
    public:
        Complex(double r, double i)   { real = r; image = i; }    // Constructor
        Complex operator + (Complex& op);                         // Method #1
        friend Complex operator - (Complex& op1, Complex& op2);   // Method #2
};

Complex Complex::operator + (Complex& op) {              // Complex + Complex
    return Complex(real + op.real, image + op.image);
}

Complex operator - (Complex& op1, Complex& op2) {        // Complex - Complex
    // Complex class에서 friend로 선언된 함수이기 때문에 private 멤버에 접근 가능
    return Complex(op1.real - op2.real, op1.image - op2.image);
}
```

Binary Operator Overloading (3)

- Member function을 이용한 operator overloading
  - operator + 함수를 호출하는 것과 같다.



- Global function과 **friend** 키워드를 이용한 operator overloading
  - 두 개의 피연산자가 각각 parameter가 되어 global function을 호출한다.



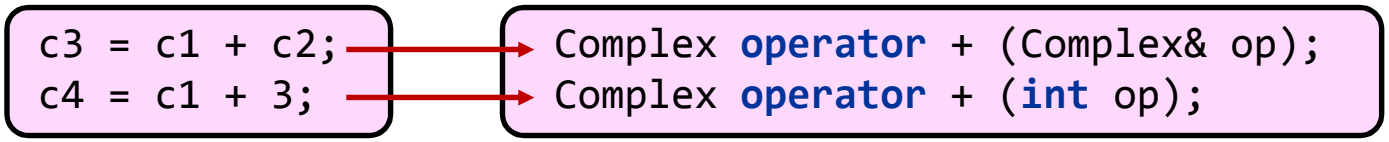
Unary Operator Overloading

- 피연산자를 하나만 가지는 연산자를 **unary operator** 라 한다.
  - ~ , ! , ++ , -- 등

```
class Complex {
    // 생략
    public:
        Complex operator ~ ();                          // Method #1
        friend Complex operator ++ (Complex& op);       // Method #2
};

Complex Complex::operator ~ ()       // ~(a + bi) = (-a + -bi)
{   return Complex(-1 * real, -1 * image);  }

Complex operator ++ (Complex& op) {  // (a + bi)++ = ((a+1) + (b+1)i)
    return Complex(op.real + 1, op.image + 1);
}
```

Miscellaneous

- 같은 연산자에 대해서 서로 다른 **type**의 피연산자 각각에 대한 operator overloading 함수를 정의할 수 있다.



- **ostream class**의 **<<** 연산자를 overloading함으로써 cout object를 이용한 화면 출력을 간편하게 할 수 있다.

```
class Complex {
    friend ostream& operator << (ostream& os, Complex& op);
};

ostream& operator << (ostream& os, Complex& op) {
    os << "( " << op.real << " + " << op.image << "i )" << endl;
    return os;                          // (real + image i) 형태로 출력
}
```

Sample Practice (1)

- **Complex** class를 구현한다.
  - `double real;`
  - `double image;`
  - `Complex(double r, double i)  // Constructor`

- `+, -, ==` 연산자에 대해 **operator overloading** 함수를 구현한다.
  - `==` 연산자 : 두 개의 `Complex` 값을 비교하여 `real`과 `image`가 모두 **같으면 1을, 그렇지 않다면 0**을 return한다.

Sample Practice (2)

- **ostream class**의 **<<** 연산자를 **overloading**하여 cout 객체를 통해 Complex 객체를 간편하게 출력할 수 있도록 한다.
  - cout << C1 과 같은 표현만으로 Complex 객체를 출력할 수 있도록 한다.
  - (*real* + *image* i) 의 형태로 출력한다.
  - **friend** 키워드를 사용해야 한다.

Sample Practice (3)

```
void main( )
{
    Complex C1(1.15, 2.45);
    Complex C2(2.12, 6.2);
    Complex C3 = C1 + C2;           // operator +
    Complex C4 = C2 - C1;           // operator -
    Complex C5 = C1 + C2 - C3 + C4; // operator +, -

    // operator << of ostream class
    cout << "C1 = " << C1 << endl;
    cout << "C2 = " << C2 << endl;
    cout << "C3 = " << C3 << endl;
    cout << "C4 = " << C4 << endl;
    cout << "C5 = " << C5 << endl;

    // operator ==
    if(C3 == C4)
        cout << "C3 == C4" << endl;
    else
        cout << "C3 != C4" << endl;

    if(C4 == C5)
        cout << "C4 == C5" << endl;
    else
        cout << "C4 != C5" << endl;
}
```

```
C1 = ( 1.15 + 2.45i )
C2 = ( 2.12 + 6.2i )
C3 = ( 3.27 + 8.65i )
C4 = ( 0.97 + 3.75i )
C5 = ( 0.97 + 3.75i )
C3 != C4
C4 == C5
계속하려면 아무 키나 누르십시오 . . .
```