

복소수 (Complex Number)

두 개의 정수 a, b 로 만들어지는 복소수 $a+bi$ (혹은 (a,b))를 구현하는 C++ 클래스 `myComplex` 를 구현하시오. 복소수 $a+bi$ 에서 a 를 실수부(real part)라고 하고, b 를 허수부(imaginary part)라고 한다.

클래스 `myComplex` 에서 가능한 복소수 산술 연산자(arithmetic operator) 로는 $+$ (덧셈), $-$ (뺄셈), $*$ (곱셈)이 가능하며, 이 산술 연산자는 다음과 같이 복소수 간의 연산 뿐만 아니라 정수와 복소수 사이의 연산도 가능하다. 다음에서 a, b, c, d 는 각각 정수이다.

$$\begin{aligned}(a, b) + (c, d) &= (a + c, b + d) \\(a, b) - (c, d) &= (a - c, b - d) \\(a, b) * (c, d) &= (ac - bd, ad + bc) \\c + (a, b) &= (c + a, b) \\c - (a, b) &= (c - a, -b) \\c * (a, b) &= (ca, cb) \\(a, b) + c &= (a + c, b) \\(a, b) - c &= (a - c, b) \\(a, b) * c &= (ac, bc)\end{aligned}$$

클래스 `myComplex` 에서는 assignment operator $=, +=, -=, *=$ 가 가능하며, 이 연산자 또한 산술 연산자와 같이 복소수 뿐만 아니라 정수에 관해서도 연산 가능하다.

복소수 $z=(a, b)$ 에 대하여, 복소수 $(a, -b)$ 를 z 의 켤레복소수(conjugate)라고 한다, 또한 복소수 $z=(a, b)$ 의 절대값(norm) $|z|$ 은 다음과 같이 정의한다.

$$|z| = \sqrt{a^2 + b^2}$$

클래스 `myComplex` 에서 가능한 복소수 관계 연산자(relational operator)로는 $==, !=, >, >=, <, <=$ 이 있으며, 다음과 같이 정의된다.

$$\begin{aligned}(a, b) == (c, d) &\text{ iff } (a == b) \text{ and } (c == d) \\(a, b) != (c, d) &\text{ iff } (a != b) \text{ or } (c != d) \\z_1 > z_2 &\text{ iff } |z_1| > |z_2| \\z_1 >= z_2 &\text{ iff } |z_1| >= |z_2| \\z_1 < z_2 &\text{ iff } |z_1| < |z_2| \\z_1 <= z_2 &\text{ iff } |z_1| <= |z_2|\end{aligned}$$

또한 myComplex 에서 가능한 단항 연산자 (unary operator)로는 -, ~, ++, -- 가 있다. 단항 연산자 -는

$$-(a, b) = (-a, -b)$$

로 정의되며, 단항 연산자 ~ 는 다음과 같이 conjugate 를 계산하는 연산자이다.

$$\sim (a, b) = (a, -b)$$

단항 연산자 ++, -- 는 각각 복소수의 정수부를 1 증가시키거나, 1 감소시키는 연산자로서 prefix 혹은 postfix 연산이 모두 가능하다.

클래스 myComplex 에는 위에서 설명한 연산자 이외에도 입출력 연산자 (<<, >>), 생성자 (constructor), 복사 생성자(copy constructor), 변경자(mutator function), 접근자(accessor function) 등이 있다.

클래스 myComplex 를 구현한 다음 두 개의 파일 MyComplex.h, MyComplex.cpp 에 위에서 설명한 모든 연산자를 추가하여 클래스 myComplex 를 완전히 구현하여, 아래 테스트 프로그램인 TestMyComplex.cpp 가 정확하게 동작하도록 하시오.

MyComplex.h

```
#ifndef MYCOMPLEX_H
#define MYCOMPLEX_H

#include <iostream>
using namespace std;

class myComplex {
public:
    // Constructor (생성자)
    myComplex(int real = 0, int imag = 0);
    // Copy constructor (복사 생성자)
    myComplex(const myComplex& number);

    // Accessor functions (접근자)
    int getRealPart() const;
    int getImaginaryPart() const;

    // Mutator functions (변경자)
    void setRealPart(int real);
    void setImaginaryPart(int imag);
    void set(int real, int imag);

    // Overloaded binary operators
    myComplex operator +(const myComplex& number) const;
    myComplex operator +(int value) const;

    // Overloaded assignment operators
    myComplex& operator =(const myComplex& number);
    myComplex& operator =(int value);
};
```

```

// Overloading relational operators
bool operator ==(const myComplex& number) const;

// Overloaded unary operators
myComplex operator -(); // unary minus

private:
    int realPart;
    int imaginaryPart;
    int norm() const;
};

#endif

```

MyComplex.cpp

```

#include "MyComplex.h"

// Constructor
myComplex::myComplex(int real, int imag)
{
    realPart = real;
    imaginaryPart = imag;
}

// Copy constructor
myComplex::myComplex(const myComplex& number)
{
    realPart = number.realPart;
    imaginaryPart = number.imaginaryPart;
}

// Accessor functions
int myComplex::getRealPart() const
{
    return realPart;
}

int myComplex::getImaginaryPart() const
{
    return imaginaryPart;
}

// Mutator functions
void myComplex::setRealPart(int real)
{
    realPart = real;
}

void myComplex::setImaginaryPart(int imag)
{
    imaginaryPart = imag;
}

void myComplex::set(int real, int imag)
{
    realPart = real;
    imaginaryPart = imag;
}

// Overloaded binary operators
myComplex myComplex::operator +(const myComplex& number) const
{
    int newReal = realPart + number.realPart;
    int newImag = imaginaryPart + number.imaginaryPart;
    return myComplex (newReal, newImag);
}

```

```

myComplex myComplex::operator +(int value) const
{
    return myComplex(value) + (*this);
}

// Assignment operators
myComplex& myComplex::operator =(const myComplex& number)
{
    this->realPart = number.realPart;
    this->imaginaryPart = number.imaginaryPart;
    return *this;
}

myComplex& myComplex::operator =(int value)
{
    realPart = value;
    imaginaryPart = 0;
    return *this;
}

// Overloading comparison operators
bool myComplex::operator ==(const myComplex& number) const
{
    return (realPart == number.realPart) &&
           (imaginaryPart == number.imaginaryPart);
}

bool myComplex::operator >(const myComplex& number) const
{
    return norm() > number.norm();
}

// Overloaded unary operators
myComplex myComplex::operator -()          // unary minus
{
    return myComplex(-realPart, -imaginaryPart);
}

// private function
int myComplex::norm() const
{
    return realPart * realPart + imaginaryPart * imaginaryPart;
}

ostream &operator <<(ostream &outStream, const myComplex& number)
{
    outStream << "(" << number.realPart << "," << number.imaginaryPart << ")";
    return outStream;
}

istream &operator >>(istream &inStream, myComplex& number)
{
    inStream >> number.realPart >> number.imaginaryPart;
    return inStream;
}

```

TestMyComplex.cpp

```

#include <fstream>
#include <cstdlib>
#include "MyComplex.h"

void testSimpleCase();
void testDataFromFile();

```

```

void main(void)
{
    testSimpleCase();
    testDataFromFile();
}

void testSimpleCase()
{
    myComplex c0, c1(1), c2(2, 2);
    myComplex c3(c2);
    myComplex c4, c5, c6, c7, c8, c9;

    // test constructor
    cout << c0 << endl << c1 << endl << c2 << endl;
    // test copy constructor
    cout << c3 << endl;

    // test accessor function
    cout << c3 << endl;

    // test mutator function
    c3.set(3, 3);
    cout << c3 << endl;

    c3.setRealPart(4);
    cout << c3 << endl;

    c3.setImaginaryPart(4);
    cout << c3 << endl;

    // test binary operators
    c4 = c1 + c3;
    c5 = c1 - c3;
    c6 = c4 * c5;
    cout << c4 << endl << c5 << endl << c6 << endl;

    c7 = c6 + 2;
    c8 = c6 - 2;
    c9 = c6 * 2;
    cout << c7 << endl << c8 << endl << c9 << endl;

    c7 += c4;
    c8 -= c5;
    c9 *= c6;
    cout << c7 << endl << c8 << endl << c9 << endl;

    c7 += 2;
    c8 -= 2;
    c9 *= 2;
    cout << c7 << endl << c8 << endl << c9 << endl;

    // test comparison operators
    cout << (c8 != c9) << " " << (c8 == c9) << endl;
    cout << (c8 > c9) << " " << (c8 >= c9) << " " << (c8 < c9) << " " << (c8 <= c9) <<
endl;
    c7 = c8 = c9;
    cout << (c8 != c9) << " " << (c8 == c9) << endl;
    cout << (c8 > c9) << " " << (c8 >= c9) << " " << (c8 < c9) << " " << (c8 <= c9) <<
endl;

    // test prefix operators
    c7 = -myComplex(2, 3);
    c8 = (++c7) * 2;
    c9 = 2 * (c7++);
    cout << c7 << " " << c8 << " " << c9 << endl;

    c7 = ~myComplex(2, 3);
    c8 = (--c7) * 2;

```

```

    c9 = 2 * (c7--);
    cout << c7 << " " << c8 << " " << c9 << endl;

    // test expressions with myComplex numbers
    c1 = myComplex(1,2);
    c2 = myComplex(2,3);
    c3 = myComplex(4,5);
    c4 = myComplex(5,6);
    c5 = myComplex(6,7);
    c6 = 3;
    cout << -(c1*c2) - 2*c3 + ~c4 * c5 * 3 + 2 ?c6 << endl;
}

void testDataFromFile()
{
    ifstream inStream;
    int numTestCases;

    inStream.open("input.txt");
    if (inStream.fail())
    {
        cerr << "Input file opening failed.\n";
        exit(1);
    }

    inStream >> numTestCases;

    for (int i=0; i<numTestCases; i++)
    {
        myComplex c1, c2, c3, c4;
        myComplex c5, c6, c7, c8, c9;

        inStream >> c1 >> c2 >> c3 >> c4;

        cout << c1 << " " << c2 << " " << c3 << " " << c4 << endl;
        cout << (2 + c1 + 3) + (2 - c2 - 3) * (2 * c3 * 3) - (2 * c4 - 3) << endl;

        c5 = c6 = c7 = c8 = c1;
        cout << (c5 == c2) << " " << (c5 != c2) << endl;
        cout << (c5 > c2) << " " << (c5 >= c2) << " " << (c5 < c2) << " " << (c5 <= c2)
<< endl;
        cout << (c5 == c6) << " " << (c5 != c6) << endl;
        cout << (c5 > c6) << " " << (c5 >= c6) << " " << (c5 < c6) << " " << (c5 <= c6)
<< endl;

        c5 += c2;
        c6 -= c3;
        c7 *= c4;
        c8 = c2.getRealPart();
        c9 = c3.getImaginaryPart();
        cout << c5 << " " << c6 << " " << c7 << " " << c8 << " " << c9 << endl;

        c7 = -c6;
        c8 = (++c7) * 2;
        c9 = 2 * (c7++);
        cout << c7 << " " << c8 << " " << c9 << endl;

        c7 = ~c6;
        c8 = (++c7) * 2;
        c9 = 2 * (c7++);
        cout << c7 << " " << c8 << " " << c9 << endl;
    }

    inStream.close();
}

```

입력

입력 파일의 이름은 “input.txt” 이다. 입력은 t 개의 테스트 케이스로 주어진다. 입력 파일의 첫 번째 줄에 테스트 케이스의 개수를 나타내는 정수 t 가 주어진다. 두 번째 줄부터 t 개의 줄에는 한 줄에 한 개의 테스트 케이스에 해당하는 네 개의 복소수를 나타내는 여덟 개의 정수가 입력된다. 각 정수들 사이에는 한 개의 공백이 있으며, 잘못된 데이터가 입력되는 경우는 없다.

출력

출력은 표준출력(standard output)을 사용한다. 출력은 테스트 프로그램인 testMyComplex.cpp 를 수행시키면 자동적으로 출력된다.

입력과 출력의 예

입력	출력
2 1 2 3 4 5 6 7 8 3 6 6 5 4 4 5 6	(0,0) (1,0) (2,2) (2,2) (2,2) (3,3) (4,3) (4,4) (5,4) (-3,-4) (1,-32) (3,-32) (-1,-32) (2,-64) (8,-28) (2,-28) (-2046,-128) (10,-28) (0,-28) (-4092,-256) 1 0 0 0 1 1 0 1 0 1 0 1 (0,-3) (-2,-6) (-2,-6) (0,-3) (2,-6) (2,-6) (211,-20) (1,2) (3,4) (5,6) (7,8) (19,-278) 0 1 0 0 1 1 1 0 0 1 0 1 (4,6) (-4,-4) (-9,22) (3,0) (6,0) (6,4) (10,8) (10,8) (-2,4) (-6,8) (-6,8) (3,6) (6,5) (4,4) (5,6) (-47,-294) 0 1 0 0 1 1 1 0 0 1 0 1 (9,11) (-1,2) (-21,48) (6,0) (4,0) (3,-2) (4,-4) (4,-4) (1,-2) (0,-4) (0,-4)