문제 82

유리수 (Rational Number)

유리수(rational number) 는 두 개의 정수 p, q ($q \neq 0$)에 대하여 분수 p/q 로 표시되는 수를 말한다. 여기서, 정수 p 를 분자(numerator)라 부르고, 정수 q 를 분모(denominator)라 부른다. 또한 정수 p 와 q 가 서로소인 경우에 분수 p/q 를 기약분수라고 한다. 유리수 연산을 가능하게 하는 C++ 클래스 "myRational"을 작성하시오. 단, 유리수 클래스는 다음과 같은 연산이 가능하도록 만들어 져야 한다.

Constructors (생성자)

(1) myRational(long num=0, long den=1);

Default constructor 로서 분수의 분모를 0 으로, 분자를 1 로 초기화하여, 분수의 초기값이 0 이 되도록 한다.

(2) myRational(const myRational &rat);

Copy constructor 이다.

Accessor Functions (접근자)

(1) long getNumerator() const;

기약분수의 분자를 리턴한다.

(2) long getDenominator() const;

기약분수의 분모를 리턴한다.

Member Functions

(1) myRational reciprocal() const;

분수의 역수(기약분수)를 리턴한다.

Overloaded Operators

myRational 에서 가능한 분수 연산자는 다음과 같다.

(4) 분수와 분수끼리의 사칙연산 (+ , -, *, /)

분수+ 분수, 분수-분수, 분수*분수, 분수/분수 연산이 가능하다. 단, 나눗셈에서 0 인 분수와의 나눗셈은 0 으로 정의한다.

(5) 분수와 정수와의 사칙연산 (+ , -, *, /)

분수+ 정수, 분수-정수, 분수*정수, 분수/정수, 정수+ 분수, 정수-분수, 정수*분수, 정수/분수 연산이 가능하다. 단, 나눗셈에서 0 인 분수 혹은 정수로 나누는 연산의 결과는 0 으로 정의한다.

(6) ++ , --

분수를 1 증가시키거나, 1 감소시킨다. Prefix, Postfix 연산이 가능하다.

(7) Unary -

분수의 부호를 반대로 한다.

(8) Comparison Operators (<, <=, >, >=, ==, !=)

비교연산자에서 분수끼리 비교하거나, 분수와 정수를 비교할 수 있다.

(9) Assignment Operators (=, + =, -=, *=, /=)

분수끼리 지정하거나 정수를 지정할 수 있다. 0 에 해당되는 분수나 정수로 나누는 경우에는 그 연산결과를 0 으로 정의한다.

(10) Input, Output Opeartors (>>, <<)

클래스 myRational 를 구현한 다음 두 개의 파일 MyRational.h, MyRational.cpp 에 위에서 설명한 모든 연산자를 추가하여 클래스 myRational 를 완전히 구현하여, 아래 테스트 프로그램인 TestMyRational.cpp 가 정확하게 동작하도록 하시오. 단, 테스트 프로그램의 testDataFromFile() 함수는 입력 파일에서 유리수를 입력하여, 그 유리수들을 오름차순으로 정렬하여 출력한다.

MyRational.h

```
#ifndef _MYRATIONAL_H_
#define _MYRATIONAL_H_

#include <iostream>
using namespace std;

class myRational
{

private:
    // 분수는 항상 내부적으로 기약분수로 표현하며, 또한 항상 _den>0 이다.
    long _num;          // numerator
    long _den;          // denominator

    long gcd(long m, long n);   // 최대공약수
    void reduce();
};

#endif
```

MyRational.cpp

```
#include "MyRational.h"

ostream &operator <<(ostream &outStream, const myRational& r)
{
    if (r._num == 0)
        outStream << 0;
    else if (r._den == 1)
        outStream << r._num;
    else
        outStream << r._num << "/" << r._den << endl;

    return outStream;
}

istream &operator >>(istream &inStream, myRational& r)
{

    inStream >> r._num >> r._den;

    if (r._den == 0)
    {
        r._num = 0;
```

```
        r._den = 1;
    }
    r.reduce();

    return inStream;
}

long myRational::gcd(long m, long n)
{
    if (m == n)
        return n;
    else if (m < n)
        return gcd(m, n-m);
    else
        return gcd(m-n, n);
}

void myRational::reduce()
{
    if (_num == 0 || _den == 0)
    {
        _num = 0;
        _den = 1;
        return;
    }
    if (_den < 0)
    {
        _den *= -1;
        _num *= -1;
    }
    if (_num == 1)
        return;

    int sgn = (_num < 0 ? -1 : 1);
    long g = gcd(sgn * _num, _den);
    _num /= g;
    _den /= g;
}
```

TestMyRational.cpp

```
#include <fstream>
#include <cstdlib>
#include "myRational.h"
using namespace std;

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

void testSimpleCase()
{
    myRational frac1(2), frac2(3, 2), frac3(6, 4), frac4(12, 8), frac5, frac6, frac7;

    cout << frac1 << " " << frac2 << " " << frac3 << " "
        << frac4 << " " << frac5 << endl;
    cout << frac1.getNumerator() << " " << frac1.getDenominator() << endl;

    // Check arithmetic operators
    cout << frac1 * frac2 << " "
        << frac1 + frac3 << " "
        << frac2 - frac1 << " "
        << frac3 / frac2 << endl;

    // Check comparison operators
```

```
    cout << ((frac1 < frac2) ? 1 : 0) << " "
        << ((frac1 <= frac2) ? 1 : 0) << " "
        << ((frac1 > frac2) ? 1 : 0) << " "
        << ((frac1 >= frac2) ? 1 : 0) << " "
        << ((frac1 == frac2) ? 1 : 0) << " "
        << ((frac1 != frac2) ? 1 : 0) << " "
        << ((frac2 < frac3) ? 1 : 0) << " "
        << ((frac2 <= frac3) ? 1 : 0) << " "
        << ((frac2 > frac3) ? 1 : 0) << " "
        << ((frac2 >= frac3) ? 1 : 0) << " "
        << ((frac2 == frac3) ? 1 : 0) << " "
        << ((frac2 != frac3) ? 1 : 0) << endl;

    cout << (frac6 = frac3) << " ";
    cout << (frac6 += frac3) << " ";
    cout << (frac6 -= frac3) << " ";
    cout << (frac6 *= frac3) << " ";
    cout << (frac6 /= frac3) << endl;

    cout << -frac6 << endl;

    frac6 = (++frac4) + 2;
    frac7 = 2 + (frac4++);
    cout << frac4 << " " << frac6 << " " << frac7 << endl;

    frac6 = (--frac4) - 2;
    frac7 = 2 - (frac4--);
    cout << frac4 << " " << frac6 << " " << frac7 << endl;

    cout << 2 * frac3 << " " << frac3 * 2 << " "
        << 2 / frac3 << " " << frac3 / 2 << endl;
}

void testDataFromFile()
{

}
```

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 한 줄에 한 개의 테스트 케이스에 해당하는 데이터가 입력된다. 각 줄에서 첫 번째로 입력되는 정수 n ($1 \le n \le 100$)은 정렬하여야 할 유리수의 개수를 나타낸다. 그 다음으로는 n 개의 유리수를 나타내는 $2n$ 개의 정수가 입력된다. 이 정수들의 절대값은 10 을 초과하지 않으며, 각 정수들 사이에는 한 개의 공백이 있다. 잘못된 데이터가 입력되는 경우는 없다.

출력

출력은 표준출력(standard output)을 사용한다. 입력되는 테스트 케이스의 순서대로 다음 줄에 이어서 각 테스트 케이스의 결과를 출력한다. 각 테스트 케이스에 해당하는 출력의 첫 번째 줄에 입력되는 유리수를 오름차순으로 출력한다. 각 유리수들 사이에는 한 개의 공백을 둔다.

입력과 출력의 예

입력	출력
3 1 -6 8 4 3 -2 -3 2 6 -4 -6 -4 5 5 1 -10 2 10 -2 25 5 10 2	2 3/2 3/2 3/2 0 2 1 3 7/2 -1/2 1 0 0 1 1 0 1 0 1 0 1 1 0 3/2 3 3/2 9/4 3/2 -3/2 7/2 9/2 9/2 3/2 1/2 -1/2 3 3 4/3 3/4 -3/4 -3/2 -3/2 -3/2 3/2 -5 -5 5 5 5