```
// Sam Borick <sb205@uakron.edu>
//
//ts interace.

#ifndef RATIONAL_HPP
#define RATIONAL_HPP

#include "test.hpp"

#include <cstdlib>
#include <iosfwd>
#include <iostream>
#include <assert.h>

// Mathematical helper functions.
//
// NOTE: These are defined in rational.cpp.
int gcd(int, int);
int lcm(int, int);


// Represents a rational number. The rational numbers are the set of
// numbers that can be represented as the quotient of two integers.
struct Rational
{
  // TODO: Define the following:
  // 1. A default constructor
        int n;
        int d;
        Rational()
        :n(0), d(1) {}

  // 2. A constructor that takes an integer value
        Rational(int num)
        :n(num), d(1){}
  // 3. A constructor that takes a pair of values
        Rational(int numer, int denom)
        :n(numer), d(denom) {
        assert(d != 0);

                int gcdnum;
                if ((numer % denom) != 0){
                        gcdnum = gcd(numer, denom);
                        numer /= gcdnum;
                        denom /= gcdnum;
                        n = numer;
                        d = denom;
                }else if(numer == 0 && denom != 1){
                        d = 1;

                }else if(numer > denom && (numer % denom == 0)){
                        n = numer / denom;
                        d = 1;
                }else{
                        //do nothing
                }
        }
  // Returns the numerator.
  int num() const {
        return n;
}

  // Returns the denominator
  int den() const {
        return d;
 }

};
```



```
inline bool operator==(Rational a, Rational b){
        return (a.n == b.n && a.d == b.d);
}
inline bool operator!=(Rational a, Rational b){
        return (a.n != b.n || a.d != b.d);
}
inline bool operator < (Rational a, Rational b){
        int lcmNum = lcm(a.d, b.d);
        int aMult, bMult;
        int newAN, newBN;
        aMult = lcmNum / a.d;
        bMult = lcmNum / b.d;
        newAN = a.n * aMult;
        newBN = b.n * bMult;
        return newAN < newBN;
}
inline bool operator > (Rational a, Rational b){
         int lcmNum = lcm(a.d, b.d);
        int aMult, bMult;
        int newAN, newBN;
        aMult = lcmNum / a.d;
        bMult = lcmNum / b.d;
        newAN = a.n * aMult;
        newBN = b.n * bMult;
        return newAN > newBN;
}
inline bool operator <= (Rational a, Rational b){
        int lcmNum = lcm(a.d, b.d);
        int aMult, bMult;
        int newAN, newBN;
        aMult = lcmNum / a.d;
        bMult = lcmNum / b.d;
        newAN = a.n * aMult;
        newBN = b.n * bMult;
        return newAN <= newBN;
}
inline bool operator >= (Rational a, Rational b){
        int lcmNum = lcm(a.d, b.d);
        int aMult, bMult;
        int newAN, newBN;
        aMult = lcmNum / a.d;
        bMult = lcmNum / b.d;
        newAN = a.n * aMult;
        newBN = b.n * bMult;
        return newAN >= newBN;
}

// 3. The standard arithmetic operators
//    - r1 + r2
//    - r1 - r2
//    - r1 * r2
//    - r1 / r2
//    - r1 / r2
inline Rational operator + (Rational a, Rational b){
        int lcmNum = lcm(a.d, b.d);
        int aMult, bMult;
        int newAN, newBN;
        aMult = lcmNum / a.d;
        bMult = lcmNum / b.d;
        newAN = a.n * aMult;
        newBN = b.n * bMult;
        Rational c((newAN + newBN), (lcmNum));
        return c;
}
inline Rational operator - (Rational a, Rational b){
        int lcmNum = lcm(a.d, b.d);
        int aMult, bMult;
        int newAN, newBN;
```

```
        aMult = lcmNum / a.d;
        bMult = lcmNum / b.d;
        newAN = a.n * aMult;
        newBN = b.n * bMult;
        Rational c((newAN − newBN), (lcmNum));
        return c;
}
inline Rational operator * (Rational a, Rational b){
        Rational c((a.n * b.n), (a.d * b.d));

        return c;
}
inline Rational operator / (Rational a, Rational b){
        Rational c((a.n * b.d), (a.d * b.n)); //multiplies by the reciprocal
        return c;
}


std::ostream& operator<<(std::ostream&, Rational);
std::istream& operator>>(std::istream&, Rational&);


#endif
```



```
// Sam Borick <sb205@uakron.edu>
//
// rational.hpp: Definition of rational class and its interace.

#include "rational.hpp"

#include <iostream>


// ------------------------------------------------------------------------ //
// Helper functions

// Compute the GCD of two integer values using Euclid's algorithm.
int
gcd(int a, int b)
{
  while (b != 0) {
    int t = b;
    b = a % b;
    a = t;
  }
  return a;
}


// Compute the LCM of two integer values.
int
lcm(int a, int b)
{
  return (std::abs(a) / gcd(a, b)) * std::abs(b);
}


// ------------------------------------------------------------------------ //
// Rational implementation


// TODO: Make this print integers when the denominator is 1.
 std::ostream&
 operator<<(std::ostream& os, Rational r)
{
  if(r.den() == 1){
   return os << r.num();
}else{
  return os << r.num() << '/' << r.den();
}
}


// TODO: Make this read integer values if no '/' is given as a separator.
// You may assume that there is no space between the numerator and the
// slash. Hint, find and read the reference documentation for istream::peek().
 std::istream&
 operator>>(std::istream& is, Rational& r)
{
  int p, q;
  char c;
  is >> p;
  c = is.peek();
  if (c == '/'){
    is >> c >> q;
    if (!is)
      return is;

     // Require that the divider to be a '/'.
    if (c != '/') {
      is.setstate(std::ios::failbit);
      return is;
    }
```

```
    // Make sure that we didn't read p/0.
    if (q == 0) {
      is.setstate(std::ios::failbit);
      return is;
    }

    r = Rational(p, q);
    return is;
  }else{
    is.setstate(std::ios::failbit);
}
}
```