```
// Karthik Venkat <kv39@zips.uakron.edu>
//
// Rational.hpp: Definition of Rational class and its interace.

#ifndef RATIONAL_HPP
#define RATIONAL_HPP
#include "test.hpp"
#include <cstdlib>
#include <iosfwd>
#include <iostream>
#include <cctype> //required for isspace
// 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
{
        //Function to simplify and Normalize the rational number
        void Normalize();

        private:
        int n, d; //Numerator and denominator
        public:
        Rational ()//Default constructor
        : n(0), d(1)
        { }

        Rational (int x)//Constructor with numerator parameter
        : n(x), d(1)
        { }

         //Constructor with numerator & denominator parameter
        Rational (int p, int q)
        : n(p), d(q)
        {
                assert(d != 0); //Asserts that denominator cannot be equal to 0.
                Normalize();    //Normalizes the values to lowest form
        }

        int num() const //Returns numerator value
        {
                return n;
        }
        int den() const //Returns denominator value
        {
         return d;
        }
};
// TODO: Implement support for
// 1. Comparing two rational numbers for equality:
// For equals operaton
bool operator == (Rational, Rational);
bool operator == (int, Rational);
bool operator == (Rational, int);
// For not equal operator
bool operator != (Rational, Rational);
bool operator != (int, Rational);
bool operator != (Rational, int);
// For less than
```



```
bool operator < (Rational, Rational);
bool operator < (int, Rational);
bool operator < (Rational, int);
//For greater than
bool operator > (Rational, Rational);
bool operator > (int, Rational);
bool operator > (Rational, int);
//For less than or equal to
bool operator <= (Rational, Rational);
bool operator <= (int, Rational);
bool operator <= (Rational, int);
//For greater than or equal to
bool operator >= (Rational, Rational);
bool operator >= (int, Rational);
bool operator >= (Rational, int);
//For addition
Rational operator + (Rational, Rational);
Rational operator + (int, Rational);
Rational operator + (Rational, int);
//For subtraction
Rational operator - (Rational, Rational);
Rational operator - (int, Rational);
Rational operator - (Rational, int);
//For Multiplication
Rational operator * (Rational, Rational);
Rational operator * (int, Rational);
Rational operator * (Rational, int);
// For division
Rational operator / (Rational, Rational);
Rational operator / (int, Rational);
Rational operator / (Rational, int);
// These are provided for you.
// NOTE: They are defined in rational.cpp.
std::ostream& operator << (std::ostream&, Rational);
std::istream& operator >> (std::istream&, Rational&);
#endif
```

```
// Karthik Venkat <kv39@zips.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);
}
//this function allows us to get the sign of the rational.
//int sign(int a) { return a < 0? -1: 1; }
// ------------------------------------------------------------------------ //
void Rational::Normalize()
{
   //Declaring variables
        assert(d != 0); //Asserting that denominator cannot be 0.
  int GCD = gcd(n, d);
  n = n / GCD;
  d = d / GCD;
        if (n == 0) d = 1; //checks if denominator is 1
  if (d < 0) //Normalizes when denominator is negative
  {
    n = n * -1;
    d = d * -1;
  }
}
//Checks for equality
bool operator == (Rational a, Rational b)
{
  return (a.num() * b.den() == a.den() * b.num());
}
bool operator == (int x, Rational r)
{
  return (r == Rational(x));
}
bool operator == (Rational r, int x)
{
  return (Rational(x) == r);
}

//Checks for inequality
bool operator != (Rational a, Rational b)
{
  return !(a.num() == b.num() && a.den() == b.den());
}
bool operator != (int x, Rational r)
{
  return !(r == Rational(x));
```



```
}
bool operator != (Rational r, int x)
{
  return !(Rational(x) == r);
}
//Checks for less than
bool operator < (Rational a, Rational b)
{
  return ((a.num() * b.den()) < (a.den() * b.num()));
}
bool operator < (int x, Rational r)
{
  return (r < Rational(x));
}
bool operator<(Rational r, int x)
{
  return (Rational(x) < r);
}
//Checks for less than or equal to
bool operator <= (Rational a, Rational b)
{
  return ((a.num() * b.den()) <= (a.den() * b.num()));
}
bool operator <= (int x, Rational r)
{
  return (r <= Rational(x));
}
bool operator <= (Rational r, int x)
{
   return (Rational(x) <= r);
}
//Chekcs for greater than or equal to
bool operator >= (Rational a, Rational b)
{
  return ((a.num() * b.den()) >= (a.den() * b.num()));
}
bool operator >= (int x, Rational r)
{
 return (r >= Rational(x));
}
bool operator >= (Rational r, int x)
{
  return (Rational(x) >= r);
}
//Checks for greater than
bool operator > (Rational a, Rational b)
{
  return ((a.num() * b.den()) > (a.den() * b.num()));
}
bool operator > (int x, Rational r)
{
  return (r > Rational(x));
}
bool operator > (Rational r, int x)
{
  return (Rational(x) > r);
}
//For the addition operation on rational numbers
Rational operator + (Rational a, Rational b)
{
  return Rational ((a.num() * b.den()) + (b.num() *
                    a.den()), (a.den() * b.den())); //Split line for neatne
```

```
}
Rational operator + (int x, Rational r)
{
  return (r + Rational(x));
}
Rational operator + (Rational r, int x)
{
  return (Rational(x) + r);
}
//For subtraction operation on rational numbers
Rational operator - (Rational a, Rational b)
{
  return Rational ((a.num() * b.den()) - (b.num() *
                   a.den()), (a.den() * b.den())); //Split line for neatness
}

Rational operator - (int x, Rational r)
{
  return (r - Rational(x));
}

Rational operator - (Rational r, int x)
{
  return (Rational(x) - r);
}

//for multiplication operations on rational numbers
Rational operator * (Rational a, Rational b)
{
  return Rational((a.num() * b.num()), (a.den() * b.den()));
}

Rational operator * (int x, Rational r)
{
  return (r * Rational(x));
}

Rational operator * (Rational r, int x)
{
  return (Rational(x) * r);
}
//For division operations on rational numbers
Rational operator / (Rational a, Rational b)
{
  return Rational((a.num() * b.den()), (a.den() * b.num()));
}

Rational operator / (int x, Rational r)
{
  return (r / Rational(x));
}

Rational operator / (Rational r, int x)
{
  return (Rational(x) / r);
}

std::ostream& operator << (std::ostream& os, Rational r)
{
  if (r.den() == 1 || r.num() == 0) //prints only relevant value
    return os << r.num();
  else
```



```
    return os << r.num() << '/' << r.den();
}

std::istream& operator >> (std::istream& in, Rational& r)
{
  int p, q;
  char c;
  q = 1;
  in >> p;
  if (in) //when input is entered, peek for operator
  {
    if (isspace(in.peek()))
    {
      //checks for white space.
      r = Rational(p, q);
      return in; //set denominator to 1
  }
  in >> c >> q;
  if (c == '/' && in && q != 0)
  {
    r = Rational(p, q);
    return in;
  }
 }
 in.setstate(std::ios::failbit);
 return in;
}
```

```
// Karthik Venkat <kv39@zips.uakron.edu>

// main.cpp: rational number test suite
#include "rational.hpp"
#include <iostream>
#include <iomanip>
#include <unistd.h>
int main()
{
  // Determine if input is coming from a terminal.
  bool term = isatty(0);
  // This will continue reading until it reaches the end-of-input.
  // If you are using this interactivly, type crtl-d to send the
  // end of input character to the terminal.
  while (std::cin)
  {
    Rational r1;
    Rational r2;
    std::string op;
    if (term)
    std::cout << "> ";
    std::cin >> r1 >> op >> r2;
    if (!std::cin)
    break;

    //Check for operators that may be used
    if (op == "==")
    std::cout << std::boolalpha << (r1 == r2) << '\n';
    else if (op == "!=")
    std::cout << std::boolalpha << (r1 != r2) << '\n';
    else if (op == "<")
    std::cout << std::boolalpha << (r1 < r2) << '\n';
    else if (op == ">")
    std::cout << std::boolalpha << (r1 > r2) << '\n';
    else if (op == "+")
    std::cout << std::boolalpha << (r1 + r2) << '\n';
    else if (op == "-")
    std::cout << std::boolalpha << (r1 - r2) << '\n';
    else if (op == "*")
    std::cout << std::boolalpha << (r1 * r2) << '\n';
    else if (op == "/")
    std::cout << std::boolalpha << (r1 / r2) << '\n';
    else if (op == "<=")
    std::cout << std::boolalpha << (r1 <= r2) << '\n';
    else if (op == ">=")
    std::cout << std::boolalpha << (r1 >= r2) << '\n';
    else
    std::cerr << "invalid operator: " << op << '\n';
  }
  // If we got to the end of the file without fatal errors,
  // return success.
  if (std::cin.eof())
  return 0;
  // Otherwise, diagnose errors in input and exit with an error
  // code.
  if (std::cin.fail())
  {
    std::cerr << "input error\n";
    return 1;
  }
  return 0;
}
```