//\*Rational Number Class Header\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

// Chris Dang CSCI 1107

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

#include <iostream>

#include <sstream>

#include <string>

#include <cctype>

using namespace std ;

class RationalNumber

{

public:

//get denominator and numerator, respectively

int getBot() ;

int getTop() ;

RationalNumber invert() ;

int gcd() ;

RationalNumber reduce() ;

int lcd(RationalNumber &ratNum2) ;

RationalNumber operator+(RationalNumber &ratNum2) ;

RationalNumber operator-(RationalNumber &ratNum2) ;

RationalNumber operator\*(RationalNumber &ratNum2) ;

RationalNumber operator/(RationalNumber &ratNum2) ;

//default constructor

RationalNumber(){numerator = 1; denominator = 2;} ;

//explicit value constructor

RationalNumber(int num, int den){numerator = num, denominator= den;} ;

void display(ostream &out) const ;

RationalNumber read(istream &in) ;

private:

int numerator, denominator ;

}; // end Rational Number class

//non-member functions

ostream &operator<<(ostream &out, const RationalNumber &Rational) ;

istream &operator>>(istream &in, RationalNumber &Rational) ;

//\*Rational Number Class Implment\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

// Chris Dang CSCI 1107

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

//#include <iostream>

using namespace std ;

#include "RationalNumberHeader.h"

int RationalNumber::getTop() {

return numerator ;

}// end getBot ;

int RationalNumber::getBot() {

return denominator ;

}// end getBot ;

RationalNumber RationalNumber::invert() {

RationalNumber ratNum(denominator, numerator) ;

return ratNum ;

}

int RationalNumber::gcd() {

//Greatest Common Divider function (GCD)

int remainder, dividend, divisor ;

if (numerator > denominator) {

dividend = numerator ;

divisor = denominator ;

}//end if

if (numerator < denominator) {

dividend = denominator ;

divisor = numerator ;

}//end if

while (divisor != 0) {

remainder = dividend % divisor ;

dividend = divisor ;

divisor = remainder ;

}// end while

return dividend ;

}//end greatestCommonDiv

RationalNumber RationalNumber::reduce() {

int div = gcd() ;

numerator = numerator / div ;

denominator = denominator / div ;

RationalNumber ratNum(numerator, denominator) ;

return ratNum ;

}// end reduce

int RationalNumber::lcd(RationalNumber &ratNum2) {

//Lowest Common Denominator function (LCD)

int store1, store2, lowComBot ;

int temp1 = getBot() ; //Get denominators of the two

int temp2 = ratNum2.getBot() ; //rational numbers so the GCD can be

//found. Once found, the product of

RationalNumber tempRat(temp1, temp2) ;//the denominators is divided by

store1 = tempRat.gcd() ; //the GCD to find the LCD

store2 = temp1 \* temp2 ;

lowComBot = store2 / store1 ;

return lowComBot ;

}// end lowestCommonBot

RationalNumber RationalNumber::operator+(RationalNumber &ratNum2) {

int numer1 = numerator ;

int numer2 = ratNum2.getTop() ;

int denom1 = denominator ;

int denom2 = ratNum2.getBot() ;

int top1 = denom1 \* numer2 ;

int top2 = denom2 \* numer1 ;

numerator = top1 + top2 ;

denominator = denom1 \* denom2 ;

RationalNumber ratNum(numerator, denominator) ;

return ratNum.reduce() ;

}//\_\_\_\_\_\_\_\_\_end overloaded + op\_\_\_\_\_\_\_\_\_\_

RationalNumber RationalNumber::operator-(RationalNumber &ratNum2) {

int numer1 = numerator ;

int numer2 = ratNum2.getTop() ;

int denom1 = denominator ;

int denom2 = ratNum2.getBot() ;

int top1 = denom1 \* numer2 ;

int top2 = denom2 \* numer1 ;

numerator = top2 - top1 ;

denominator = denom1 \* denom2 ;

RationalNumber ratNum(numerator, denominator) ;

return ratNum.reduce() ;

}//\_\_\_\_\_\_\_\_\_end overloaded - op\_\_\_\_\_\_\_\_\_\_

RationalNumber RationalNumber::operator\*(RationalNumber &ratNum2) {

numerator = numerator \* ratNum2.getTop() ;

denominator = denominator \* ratNum2.getBot() ;

RationalNumber ratNum(numerator, denominator) ;

return ratNum.reduce() ;

}//\_\_\_\_\_\_\_\_\_end overloaded \* op\_\_\_\_\_\_\_\_\_\_

RationalNumber RationalNumber::operator/(RationalNumber &ratNum2) {

ratNum2 = ratNum2.invert() ;

numerator = numerator \* ratNum2.getTop() ;

denominator = denominator \* ratNum2.getBot() ;

RationalNumber ratNum(numerator, denominator) ;

return ratNum.reduce() ;

}//\_\_\_\_\_\_\_\_\_end overloaded / op\_\_\_\_\_\_\_\_\_\_

void RationalNumber::display(ostream &out) const{

if (denominator != 1 )

out << numerator << '/' << denominator ;

else

out << numerator ;

}//end display

RationalNumber RationalNumber::read (istream &in) {

int num1, num2 ;

char ch ;

//string temp = "";

in >> num1 ;

if (in.peek() == '/') {

in >> ch ;

if(isdigit(in.peek()))

in >> num2 ;

}// end if

if(num2 == 0) {

cout << "Error: Denominator cannot be zero" << endl <<

"Terminating Program" ;

exit(1107) ;

}//end if

numerator = num1 ;

denominator = num2 ;

RationalNumber ratNum(numerator, denominator) ;

return ratNum ;

}//end read function

//----------non-member functions --------------------------------------

ostream &operator<<(ostream &out, const RationalNumber &Rational){

Rational.display(out);

return out;

}// end overloaded insertion operator

istream &operator>>(istream &in, RationalNumber &Rational) {

Rational = Rational.read(in) ;

return in ;

}// end overloaded extraction operator

/\*--------------------------------------------------------------------

A rational number calculator.

Input: rational numbers and operators

Output: messages and results of operations on rational numbers

Modified by Chris Dang 2/22/2015

--------------------------------------------------------------------\*/

#include <iostream> // cin, cout, >>, <<

#include "RationalNumberHeader.h" // Rational

using namespace std;

int main()

{

const char HELP\_INFO[] =

"\nA rational number x can be entered in the form"

"\na or a/b. (e.g. 4, 3/4, 5/2)"

"\n"

"\nValid inputs:"

"\n x + y addition (lowest terms)"

"\n x - y subtraction (lowest terms)"

"\n x \* y multiplication (lowest terms)"

"\n x / y division (lowest terms)"

"\n x I inverse of x"

"\n x M x as a mixed number"

"\n x R x reduced to lowest terms"

"\n x G greatest common divisor"

"\n x L y lowest common denominator";

const char BAD\_INPUT[] = "\*\*\* Bad input! Type 'h' for help. \*\*\*\n";

cout << "\nThis program is a rational number calculator.\n";

for (;;)

{

cout << "\nInput (h=help,q=quit): ";

if (cin.peek() == 'h')

{

cin.get();

cout << HELP\_INFO << endl;

}

else if (cin.peek() == 'q')

{

cin.get();

return 0;

}

else

{

RationalNumber x, y;

char op;

cin >> x >> op;

switch (op)

{

case '+':

cin >> y;

cout << x + y << endl;

break;

case '-':

cin >> y;

cout << x - y << endl;

break;

case '\*':

cin >> y;

cout << x \* y << endl;

break;

case '/':

cin >> y;

cout << x / y << endl;

break;

case 'i': case 'I':

cout << x.invert() << endl;

break;

case 'r': case 'R':

x.reduce();

cout << x << endl;

break;

case 'g': case 'G':

cout << x.gcd() << endl;

break;

case 'l': case 'L':

cin >> y;

cout << x.lcd(y) << endl;

break;

default:

cout << BAD\_INPUT;

}

}

if (cin.fail())

{

cout << BAD\_INPUT;

cin.clear();

}

cin.ignore(9999,'\n');

}

}

