COMSATS Institute of Information Technology, Vehari

Department of Computer Science

Lab # 4

Solution

Subject: CSC241 Object Oriented Programming Instructor: Saif Ullah Ijaz

Exercise:

Create a class **`RationalNumber`** (fractions) with the following capabilities:

a) Create a constructor that prevents a 0 denominator in a fraction, reduces or simplifies fractions that are not in reduced form and avoids negative denominators.
b) Overload the addition, subtraction, multiplication and division operators for this class.
c) Overload the input and output stream operators for this class.
d) Overload the relational and equality operators for this class.

Solution:

RationalNumber.h

```
#include<iostream>
using namespace std;

class RationalNumber{
	int numerator, denominator;
	void ReducedForm(RationalNumber &); //Helper function to convert a rational into
reduced form
public:
	RationalNumber(int = 0, int = 1);

	RationalNumber & operator +=(const RationalNumber &);
	RationalNumber & operator +=(const int);
	RationalNumber & operator -=(const RationalNumber &);
	RationalNumber & operator -=(const int);
	RationalNumber & operator *=(const RationalNumber &);
	RationalNumber & operator *=(const int);
	RationalNumber & operator /=(const RationalNumber &);
	RationalNumber & operator /=(const int);

	friend ostream & operator << (ostream & , const RationalNumber &);
	friend istream & operator >> (istream & , RationalNumber &);

	bool operator == (const RationalNumber &);
	bool operator != (const RationalNumber &);
	bool operator < (const RationalNumber &);
	bool operator > (const RationalNumber &);
	bool operator <= (const RationalNumber &);
	bool operator >= (const RationalNumber &);
};
```

RationalNumber.cpp

```cpp
#include<algorithm>          // Library required to use the minimum (min) function
#include<cmath>
#include "RationalNumber.h"

RationalNumber::RationalNumber(int num, int den) :numerator(num), denominator(den){
	if (denominator <= 0)
		denominator = 1;
	ReducedForm(*this);

}

void RationalNumber::ReducedForm(RationalNumber &obj){
	int Divisor = std::min(abs(obj.numerator), abs(obj.denominator));
	for (int d = Divisor; d > 1; d--){
		while ((obj.numerator%d == 0) && (obj.denominator%d == 0)){
			obj.numerator /= d;
			obj.denominator /= d;
		}
	}
}

RationalNumber & RationalNumber::operator +=(const RationalNumber & rhs){
	numerator = (numerator*rhs.denominator) + (rhs.numerator*denominator);
	denominator = denominator*rhs.denominator;
	ReducedForm(*this);
	return *this;
}

RationalNumber & RationalNumber::operator +=(const int rhs){
	numerator = numerator + (rhs*denominator);
	ReducedForm(*this);
	return *this;
}

RationalNumber & RationalNumber::operator -=(const RationalNumber & rhs){
	numerator = (numerator*rhs.denominator) - (rhs.numerator*denominator);
	denominator = denominator*rhs.denominator;
	ReducedForm(*this);
	return *this;
}

RationalNumber & RationalNumber::operator -=(const int rhs){
	numerator = numerator - (rhs*denominator);
	ReducedForm(*this);
	return *this;
}

RationalNumber & RationalNumber::operator *=(const RationalNumber & rhs){
	numerator = numerator*rhs.numerator;
	denominator = denominator*rhs.denominator;
	ReducedForm(*this);
	return *this;
}

RationalNumber & RationalNumber::operator *=(const int rhs){
	numerator = numerator*rhs;
	ReducedForm(*this);
	return *this;
}

RationalNumber & RationalNumber::operator /=(const RationalNumber & rhs){
	numerator = numerator*rhs.denominator;
	denominator = denominator*rhs.numerator;
```

```
        ReducedForm(*this);
        return *this;
}
RationalNumber & RationalNumber::operator /=(const int rhs){
        denominator = denominator*rhs;
        ReducedForm(*this);
        return *this;
}

bool RationalNumber::operator ==(const RationalNumber & r){
        if (((float)numerator / (float)denominator) == ((float)r.numerator /
(float)r.denominator))
                return true;
        else
                return false;
}

bool RationalNumber::operator !=(const RationalNumber & r){
        if (((float)numerator / (float)denominator) != ((float)r.numerator /
(float)r.denominator))
                return true;
        else
                return false;
}

bool RationalNumber::operator <(const RationalNumber & r){
        if (((float)numerator / (float)denominator) < ((float)r.numerator /
(float)r.denominator))
                return true;
        else
                return false;
}

bool RationalNumber::operator >(const RationalNumber & r){
        if (((float)numerator / (float)denominator) > ((float)r.numerator /
(float)r.denominator))
                return true;
        else
                return false;
}

bool RationalNumber::operator <=(const RationalNumber & r){
        if ((*this<r) || (*this==r))
                return true;
        else
                return false;
}

bool RationalNumber::operator >=(const RationalNumber & r){
        if ((*this>r) || (*this == r))
                return true;
        else
                return false;
}
```

Source.cpp

```
#include "RationalNumber.h"

RationalNumber operator +(const RationalNumber &, const RationalNumber &);
RationalNumber operator +(const int, const RationalNumber &);
RationalNumber operator +(const RationalNumber &, const int);
RationalNumber operator -(const RationalNumber &, const RationalNumber &);
RationalNumber operator -(const int, const RationalNumber &);
RationalNumber operator -(const RationalNumber &, const int);
RationalNumber operator *(const RationalNumber &, const RationalNumber &);
RationalNumber operator *(const int, const RationalNumber &);
RationalNumber operator *(const RationalNumber &, const int);
RationalNumber operator /(const RationalNumber &, const RationalNumber &);
RationalNumber operator /(const int, const RationalNumber &);
RationalNumber operator /(const RationalNumber &, const int);

void main(){

	RationalNumber r1(4, 9), r2(2,3), r3;

	cin >> r1 >> r2 >> r3;
	cout << "First Rational Number: " << r1 << endl;
	cout << "Second Rational Number: " << r2 << endl;
	cout << "Third Rational Number: " << r3 << endl << endl;


	cout << "Addition: " << r1 + r2 + r3 << endl;
	cout << "Subtraction: " << r1 - r2 - r3 << endl;
	cout << "Multiplication: " << r1 * r2 * r3 << endl;
	cout << "Division: " << r1 / r2 / r3 << endl << endl;

	if (r1 == r2)
		cout << "r1 is equal to r2" << endl;
	if (r1 != r2)
		cout << "r1 is not equal to r2" << endl;
	if (r1 < r2)
		cout << "r1 is less than r2" << endl;
	if (r1 <= r2)
		cout << "r1 is less than or equal to r2" << endl;
	if (r1 > r2)
		cout << "r1 is greater than r2" << endl;
	if (r1 >= r2)
		cout << "r1 is greater than or equal to r2" << endl;

	system("pause");

}

ostream & operator << (ostream & os, const RationalNumber & r){
	if (r.denominator == 1)
		os << r.numerator;
	else
		os << r.numerator << " / " << r.denominator;
	return os;
}

istream & operator >> (istream & in, RationalNumber & r){
	cout << "Enter value for numerator: ";
	in >> r.numerator;
	cout << "Enter value for denominator: ";
	in >> r.denominator;
	r.ReducedForm(r);
	return in;
}
```

```
RationalNumber operator +(const RationalNumber & lhs, const RationalNumber & rhs){
	RationalNumber t = lhs;
	return t += rhs;
}

RationalNumber operator +(const int a, const RationalNumber & rhs){
	RationalNumber t = rhs;
	return t += a;
}

RationalNumber operator +(const RationalNumber & lhs, const int a){
	RationalNumber t = lhs;
	return t += a;
}

RationalNumber operator -(const RationalNumber & lhs, const RationalNumber & rhs){
	RationalNumber t = lhs;
	return t -= rhs;
}

RationalNumber operator -(const int a, const RationalNumber & rhs){
	RationalNumber t(a);
	return t -= rhs;
}

RationalNumber operator -(const RationalNumber & lhs, const int a){
	RationalNumber t = lhs;
	return t -= a;
}

RationalNumber operator *(const RationalNumber & lhs, const RationalNumber & rhs){
	RationalNumber t = lhs;
	return t *= rhs;
}

RationalNumber operator *(const int a, const RationalNumber & rhs){
	RationalNumber t(a);
	return t *= rhs;
}

RationalNumber operator *(const RationalNumber & lhs, const int a){
	RationalNumber t = lhs;
	return t *= a;
}

RationalNumber operator /(const RationalNumber & lhs, const RationalNumber & rhs){
	RationalNumber t = lhs;
	return t /= rhs;
}

RationalNumber operator /(const int a, const RationalNumber & rhs){
	RationalNumber t(a);
	return t /= rhs;
}

RationalNumber operator /(const RationalNumber & lhs, const int a){
	RationalNumber t = lhs;
	return t /= a;
}
```