ГУАП

КАФЕДРА №14

ОТЧЕТ

ЗАЩИЩЕН С ОЦЕНКОЙ

ПРЕПОДАВАТЕЛЬ

Должность, уч. степень, звание подпись, дата инициалы, фамилия

**ОТЧЕТ О ЛАБОРАТОРНОЙ РАБОТЕ №2**

по курсу: ТЕХНОЛОГИЯ ПРОГРАММИРОВАНИЯ

РАБОТУ ВЫПОЛНИЛ

СТУДЕНТ ГР. 1441 М.И. Лубинец

подпись, дата инициалы, фамилия

Санкт-Петербург

2015

1. Формализация задачи

Задача №1:

Создать класс, обеспечивающего работу с рациональными дробями. Класс должен уметь

работать как с рациональными дробями, так и с целыми числами. Должны быть определены операции

сложения, вычитания, умножения и деления. Дроби должны быть упрощены и приведены к правильным.

2. Листинги

Файл mmath.hpp

#ifndef MMATH\_H

#define MMATH\_H

#include <assert.h>

#include <type\_traits>

#include <utility>

**namespace** msvd {

**namespace** math {

// Greatest Common Divisor

**template** <**typename** T>

inline T

GCD (T gcd, T b) {

**static\_assert**(std::is\_arithmetic<T>::value,

"Arithmetic types required");

T t;

**if**(gcd < b) std::swap(gcd, b);

**while**(b) {

t = b;

b = gcd % b;

gcd = t;

}

**return** gcd;

}

}

}

#endif // MMATH\_H

Файл rational.hpp  
#ifndef RATIONAL\_H  
#define RATIONAL\_H  
#include <stdint.h>  
#include <iostream>  
#include <type\_traits>  
#include <cmath>

**namespace** msvd {

**namespace** math {

**class** Rational {

**public**:

Rational();

Rational(**int** num, **int** denum);

Rational(**const** Rational&) = **default**;

Rational& **operator**= (**const** Rational& from);

**template**<**typename** T>

Rational& **operator**= (T from) {

**static\_assert**(std::is\_arithmetic<T>::value,

"Arithmetic value is required");

**if**(std::is\_integral<T>::value) {

\_dec = from;

\_num = 1;

\_denum = 1;

} **else** {

// Extracting integral and exponent values

T i, f;

f = std::modf(from, &i);

\_denum = 1000;

\_num = **static\_cast**<**int**>(round(f \* 1000.0));

\_dec = **static\_cast**<**int**>(i);

}

Simplify();

**return** \***this**;

}

/\* Addition \*/

**friend** Rational **operator**+ (**const** Rational& a, **const** Rational& b);

**friend** Rational **operator**+ (**const** Rational& a, **int** b);

**friend** Rational **operator**+ (**int** b, **const** Rational& a);

/\* Substraction \*/

**friend** Rational **operator**- (**const** Rational& a, **const** Rational& b);

**friend** Rational **operator**- (**const** Rational& a, **int** b);

**friend** Rational **operator**- (**int** b, **const** Rational& a);

/\* Multiplication \*/

**friend** Rational **operator**\* (**const** Rational& a, **const** Rational& b);

**friend** Rational **operator**\* (**const** Rational& a, **int** b);

**friend** Rational **operator**\* (**int** b, **const** Rational& a);

/\* Division \*/

**friend** Rational **operator**/ (**const** Rational& a, **const** Rational& b);

**friend** Rational **operator**/ (**const** Rational& a, **int** b);

**friend** Rational **operator**/ (**int** b, **const** Rational& a);

/\* Streams \*/

**friend** std::ostream& **operator** << (std::ostream& os, **const** Rational& obj);

**friend** std::istream& **operator** >> (std::istream& is, Rational& obj);

/\* Getters \*/

**private**:

**void** Rationalize();

**void** Simplify();

**int** \_num; // Numenator

**int** \_denum; // Denumenator

**unsigned** \_dec; // Integer

};

} // math

} // msvd

#endif // RATIONAL\_H

Файл rational.cpp

#include "rational.hpp"

#include "mmath.hpp"

#include <cmath>

#include <utility>

#include <type\_traits>

#include <assert.h>

**namespace** msvd {

**namespace** math {

Rational::Rational() : \_num(0), \_denum(0), \_dec(0) {}

Rational::Rational(**int** num, **int** denum) : \_num(num), \_denum(denum), \_dec(0) {

Simplify();

}

**void**

Rational::Rationalize() {

\_dec += \_num / \_denum;

\_num = \_num % \_denum;

}

**void**

Rational::Simplify() {

Rationalize();

// Find greatest common divisor

**int** gcd = GCD(\_num, \_denum);

// Divide both numenator and denumenator by gcd

\_num /= gcd;

\_denum /= gcd;

}

Rational&

Rational::**operator**=(**const** Rational& from) {

\_num = from.\_num;

\_denum = from.\_denum;

\_dec = from.\_dec;

**return** \***this**;

}

Rational

**operator**+ (**const** Rational& a, **const** Rational& b) {

**int** a\_num = a.\_num + a.\_dec \* a.\_denum;

**int** b\_num = b.\_num + b.\_dec \* b.\_denum;

**int** a\_den = a.\_denum;

**int** b\_den = b.\_denum;

a\_num \*= b\_den;

b\_num \*= a\_den;

**return** Rational(a\_num + b\_num, a\_den \* b\_den);

}

Rational

**operator**+ (**const** Rational& a, **int** b) {

b \*= a.\_denum;

**return** Rational(a.\_num + b, a.\_denum);

}

Rational

**operator**+ (**int** b, **const** Rational& a) {

**return** **operator**+(a, b);

}

Rational

**operator**- (**const** Rational& a, **const** Rational& b) {

**int** a\_num = a.\_num + a.\_dec \* a.\_denum;

**int** b\_num = b.\_num + b.\_dec \* b.\_denum;

**int** a\_den = a.\_denum;

**int** b\_den = b.\_denum;

a\_num \*= b\_den;

b\_num \*= a\_den;

**return** Rational(a\_num - b\_num, a\_den \* b\_den);

}

Rational

**operator**- (**const** Rational& a, **int** b) {

b \*= a.\_denum;

**return** Rational(a.\_num + b, a.\_denum);

}

Rational

**operator**- (**int** b, **const** Rational& a) {

b \*= a.\_denum;

**return** Rational(a.\_num + b, a.\_denum);

}

Rational

**operator**\* (**const** Rational& a, **const** Rational& b) {

**int** a\_num = a.\_num + a.\_dec \* a.\_denum;

**int** b\_num = b.\_num + b.\_dec \* b.\_denum;

**return** Rational(a\_num \* b\_num, a.\_denum \* b.\_denum);

}

Rational

**operator**\* (**const** Rational& a, **int** b) {

**return** Rational(a.\_num \* b, a.\_denum);

}

Rational

**operator**\* (**int** b, **const** Rational& a) {

**return** Rational(a.\_num \* b, a.\_denum);

}

Rational

**operator**/ (**const** Rational& a, **const** Rational& b) {

**int** a\_num = a.\_num + a.\_dec \* a.\_denum;

**int** b\_num = b.\_num + b.\_dec \* b.\_denum;

**return** Rational(a\_num \* b.\_denum, b\_num \* a.\_denum);

}

Rational

**operator**/ (**const** Rational& a, **int** b) {

**return** Rational(a.\_num, a.\_denum \* b);

}

Rational

**operator**/ (**int** b, **const** Rational& a) {

**return** Rational(a.\_num, a.\_denum \* b);

}

std::ostream&

**operator**<<(std::ostream& os, **const** Rational& obj) {

**return** os << ((obj.\_dec) ? std::to\_string(obj.\_dec) + "+" : "") << obj.\_num << "/" << obj.\_denum;

}

std::istream&

**operator**>>(std::istream& is, Rational& obj) {

is >> obj.\_num >> obj.\_denum;

obj.Simplify();

**return** is;

}

} // math

} // msvd

Файл main.cpp

#include <iostream>

#include "rational.hpp"

**using** **namespace** std;

**using** **namespace** msvd::math;

**int** main(**int** argc, **char** \*argv[]) {

Rational pi(22, 7);

Rational e(8, 3);

Rational y;

**double** dfract;

std::cout << "Enter your double: ";

std::cin >> dfract;

y = dfract;

std::cout << "PI: " << pi << std::endl

<< "E: " << e << std::endl

<< "Y: " << y << std::endl << std::endl;

std::cout << "PI + E = " << pi + e << std::endl;

std::cout << "PI - E = " << pi - e << std::endl;

std::cout << "PI \* E = " << pi \* e << std::endl;

std::cout << "PI / E = " << pi / e << std::endl;

Rational d = pi \* e;

std::cout << "\nD = PI \* E = " << d << std::endl;

std::cout << "D + Y = " << d + y << std::endl;

std::cout << "D - Y = " << d - y << std::endl;

std::cout << "D \* Y = " << d \* y << std::endl;

std::cout << "D / Y = " << d / y << std::endl;

**return** 0;

}

3. Примеры

Вводимая десятичная дробь: 5.55 (Далее Y)

PI: 3+1/7

E: 2+2/3

Y: 5+11/20

PI + E = 5+17/21

PI - E = 10/21

PI \* E = 8+8/21

PI / E = 1+5/28

D = PI \* E = 8+8/21

D + Y = 13+391/420

D - Y = 2+349/420

D \* Y = 46+18/35

D / Y = 1+1189/2331

Примечание: в дробях вида 3+1/7, 3 -- целая часть, 1/7 -- дробная.

В случае отрицательной дроби, знак “минус” применяется к дробной части.