Московский государственный технический

университет им. Н.Э. Баумана

Проверил:

Преподаватель каф. ИУ5

Подпись и дата:

Факультет «Информатика и система управления»

Кафедра ИУ5 «Система обработки информации и управления»

Выполнил:

Студент группы ИУ5-22Б

Яншин Т. И.

Подпись и дата:

Курс «Программирование на основе классов и шаблонов»

Отчёт по лабораторной работе №3

Классы. Перегрузка конструкторов и операций

## **Задание**

***Лабораторная работа состоит из двух частей:***

1. ***Часть 1:***
   1. Создать класс "Дроби" для выполнения арифметических операций над обыкновенными дробями. Внутреннее представление дробей (состав полей класса) должно обеспечивать эффективное выполнение операций над дробями и может отличаться от представления дробей на экране монитора, которое должно быть удобным для пользователя. Например, внутри объекта класса "Дроби" может храниться неправильная дробь.
   2. Разработать конструктор по умолчанию и конструктор, который преобразует строку, содержащую внешнее представление дроби, в объект класса "Дроби".
   3. Перегрузить операции потокового ввода дроби с клавиатуры и вывода ее на экран монитора. При вводе выполнять сокращение дроби.
   4. Создать многофайловый проект и отладить программу, которая создает один объект класса "Дроби" и выводит значения его полей на экран (эта программа состоит из 5 первых операторов программы, приведённой в [Приложении 1](https://iu5edu.ru/wiki/cpp2/docs/labs/lab3/Instructions/SupplementMaterial1/)).
2. ***Часть 2:***
   1. Дополнить класс функциями-членами класса и функциями-друзьями класса, которые необходимы для выполнения программы из [Приложения 1](https://iu5edu.ru/wiki/cpp2/docs/labs/lab3/Instructions/SupplementMaterial1/).
   2. Выполнить программу из [Приложения 1](https://iu5edu.ru/wiki/cpp2/docs/labs/lab3/Instructions/SupplementMaterial1/) и сравнить результаты с тестовым примером.

**Создание класса**[**​**](https://iu5edu.ru/wiki/cpp2/docs/labs/lab3/Instructions/#%D1%81%D0%BE%D0%B7%D0%B4%D0%B0%D0%BD%D0%B8%D0%B5-%D0%BA%D0%BB%D0%B0%D1%81%D1%81%D0%B0)

В данной лабораторной работе необходимо создать класс для работы с обыкновенными дробями. Все операции, которые должны выполняться с дробями, включены в программу в [приложении 1](https://iu5edu.ru/wiki/cpp2/docs/labs/lab3/Instructions/SupplementMaterial1/).

Числитель и знаменатель дроби имеют тип int.

Дроби вводятся как строка, имеющая вид:

* для дробей **с целой частью**: знак, целая часть, пробел, числитель, слэш (/), знаменатель. Например: -2 6/18, 5 9/3, 2 4/1.
* для дробей **без целой части**: знак, числитель, слэш (/), знаменатель. Например: 3/4, -9/3, -8/6 (знаменатель всегда положительный).
* для дробей **без дробной части**. Например: -2, 0, 4.

Перегрузка операции

Для работы с дробями необходимо:

Перегрузить операции +, += для сложения дробей и дроби и целого в любых сочетаниях (дробь+целое, целое+дробь, дробь+дробь).

Перегрузить операции +, += для сложения дроби и double в любых сочетаниях (дробь+double, double+дробь). Преобразование double-дробь должно выполняться с точностью до N\_DEC десятичных знаков после запятой, где N\_DEC - целочисленная константа, задаваемая пользователем. Задайте значение по умолчанию N\_DEC=4.

## **Текст программы**

Внешний CMakeLists.txt

cmake\_minimum\_required (VERSION 3.8)

set(project "lab3")

project(**${project}**)

set(CMAKE\_CXX\_STANDARD 17)

set(**${project}**\_SOURCES

main.cpp)

add\_subdirectory(fraction)

set(**${project}**\_SOURCE\_LIST

${**${project}**\_SOURCES})

add\_executable(**${project}**

${**${project}**\_SOURCE\_LIST} )

target\_link\_libraries(**${project}** fraction)

Внутренний fraction/CMakeLists.txt

cmake\_minimum\_required(VERSION 3.8)

set(project "fraction")

project(**${project}**)

set(CMAKE\_CXX\_STANDARD 17)

set(**${project}**\_SOURCES

fraction.cpp)

set(**${project}**\_HEADERS

fraction.h)

set(**${project}**\_SOURCE\_LIST

${**${project}**\_SOURCES}

${**${project}**\_HEADERS})

add\_library(**${project}**

STATIC

${**${project}**\_SOURCE\_LIST})

Main.cpp

#include <iostream>

#include <iomanip>

#include "fraction/fraction.h"

int main()

{

setlocale(LC\_ALL, "Russian");

// ввод дроби с клавиатуры

std::cout << "Введите дробь: \n";

fraction z;

std::cin >> z;

std::cout << "z = " << z << std::endl;

std::cout << "=====TEST=====" << std::endl;

// проверка конструкторов

fraction fr1(10, 14), fr2;

std::cout << "fr2 = " << fr2 << std::endl;

std::cout << "fr1 = " << fr1 << std::endl;

fraction fr = "-1 4/8";

std::cout << "fr = " << fr << std::endl;

fraction x(z), y;

std::cout << "x = " << x << std::endl;

y = x + fr;

std::cout << "Test:" << y << std::endl;

double dbl = -1.25;

fraction f = dbl;

std::cout << "f=" << f << std::endl;

// проверка перегруженной операции "+"

y = x + z;

std::cout << "y=" << y << std::endl;

y += x;

f += dbl / 2;

std::cout << "f=" << f << std::endl;

y = x + dbl;

std::cout << "y = " << y << std::endl;

y = dbl + y;

std::cout << "y = " << y << std::endl;

y += dbl;

std::cout << "y=" << y << std::endl;

int i = 5;

y += i;

std::cout << "y=" << y << std::endl;

y = i + x;

std::cout << "y=" << y << std::endl;

y = x + i;

std::cout << "y=" << y << std::endl;

y += dbl + i + x;

std::cout << "y=" << y << std::endl;

return 0;

}

Fraction.h

#include <iostream>

class fraction

{

private:

int numerator{};

int denominator{};

public:

// Fraction constructors

fraction(int first, int second);

fraction(const char\* string);

fraction(fraction& o);

fraction(double value);

fraction() : numerator(0), denominator(1) {};

// Fraction getters

int getNumerator() const;

int getDenominator() const;

// Fraction functions

friend void fractionParse(fraction& o, const char\* string);

// Fraction setters

void setNumerator(int first);

void setDenominator(int second);

// Fractions operators overload

// friend fraction operator+(int wholePart, fraction &o);

fraction operator+(fraction& o) const;

friend fraction operator+(fraction& frac, double dbl);

friend fraction operator+(double dbl, fraction& frac);

friend fraction operator+(int i, fraction& frac);

fraction operator+=(fraction& frac);

fraction operator+=(double dbl);

fraction operator+=(int i);

fraction operator=(fraction const& o);

friend std::ostream& operator<<(std::ostream& out, fraction& o);

friend std::istream& operator>>(std::istream& in, fraction& o);

};

Fraction.cpp

#include "fraction.h"

fraction::fraction(int first, int second) : numerator(first), denominator(second)

{

int commonDivisor = std::gcd(first, second);

first /= commonDivisor;

second /= commonDivisor;

this->numerator = first;

this->denominator = second;

}

fraction::fraction(const char\* string) {

fractionParse(\*this, string);

}

fraction::fraction(double value)

{

static const int N\_DEC = 4;

int integerPart = static\_cast<int>(value);

double decimalPart = value - integerPart;

// Преобразование дробной части в дробь

double numeratorDouble = decimalPart \* pow(10, N\_DEC);

int commonFactor = std::gcd(static\_cast<int>(numeratorDouble), static\_cast<int>(pow(10, N\_DEC)));

numerator = static\_cast<int>(numeratorDouble / commonFactor);

denominator = static\_cast<int>(pow(10, N\_DEC) / commonFactor);

// Добавление целой части к дроби

numerator += integerPart \* denominator;

// Сокращение дроби

int divisor = std::gcd(numerator, denominator);

numerator /= divisor;

denominator /= divisor;

}

int fraction::getNumerator() const {

return numerator;

}

int fraction::getDenominator() const {

return denominator;

}

void fraction::setNumerator(int first) {

this->numerator = first;

}

void fraction::setDenominator(int second) {

this->denominator = second;

}

std::ostream& operator<<(std::ostream& out, fraction& o)

{

int tempNumerator = o.numerator;

int tempDenominator = o.denominator;

if (abs(tempNumerator) > tempDenominator && tempDenominator != 0)

{

int commonDivisor = std::gcd(tempNumerator, tempDenominator);

tempNumerator /= commonDivisor;

tempDenominator /= commonDivisor;

int wholePart = tempNumerator / tempDenominator;

tempNumerator = tempNumerator - wholePart \* tempDenominator;

if (tempNumerator < 0)

{

tempNumerator \*= -1;

out << wholePart << " " << tempNumerator << '/' << tempDenominator;

return out;

}

else if (tempNumerator == 0)

{

out << wholePart;

return out;

}

else

{

out << wholePart << " " << tempNumerator << '/' << tempDenominator;

return out;

}

}

else if (abs(tempNumerator) == tempDenominator)

{

if (tempNumerator < 0) {

out << "-1";

return out;

}

else {

out << "1";

return out;

}

}

else if (tempDenominator != 0 && tempNumerator != 0) {

out << tempNumerator << '/' << tempDenominator;

return out;

}

else if (tempDenominator != 0 && tempNumerator == 0) {

out << tempNumerator;

return out;

}

else {

out << "Неправильный ввод!";

return out;

}

}

bool find(const char\* string)

{

int i = 0;

while (string[i] != '\0')

{

if (string[i] == ' ')

{

return true;

}

i++;

}

return false;

}

std::istream& operator>>(std::istream& in, fraction& o) {

char\* string = new char[45] {};

in.getline(string, 45);

fractionParse(o, string);

delete[] string;

return in;

}

void fractionParse(fraction& o, const char\* string)

{

char\* wholePart = new char[45] {};

char\* numeratorChar = new char[45] {};

char\* denominatorChar = new char[45] {};

bool ceil = 1;

bool under = 0;

if (string[0] == '-')

{

under = 1;

}

if (find(string))

{

int i = 0;

int index = 0;

while (string[i] != ' ')

{

wholePart[index] = string[i];

index++;

i++;

}

i++;

index = 0;

while (string[i] != '/')

{

numeratorChar[index] = string[i];

index++;

i++;

}

i++;

index = 0;

while (string[i] != '\0') {

denominatorChar[index] = string[i];

index++;

i++;

}

if (atoi(wholePart) < 0)

{

o.numerator = -1 \* atoi(numeratorChar) + atoi(denominatorChar) \* atoi(wholePart);

o.denominator = atoi(denominatorChar);

}

else if (under == 1)

{

o.numerator = -1 \* atoi(numeratorChar) + atoi(denominatorChar) \* atoi(wholePart);

o.denominator = atoi(denominatorChar);

}

else

{

o.numerator = atoi(numeratorChar) + atoi(denominatorChar) \* atoi(wholePart);

o.denominator = atoi(denominatorChar);

}

}

else

{

int index = 0;

int i = 0;

while (string[i] != '/' && string[i] != '\0')

{

numeratorChar[index] = string[i];

index++;

i++;

}

if (string[i + 1] == '/')

{

ceil = 0;

}

index = 0;

i++;

while (string[i] != '\0')

{

denominatorChar[index] = string[i];

index++;

i++;

}

o.numerator = atoi(numeratorChar);

o.denominator = atoi(denominatorChar);

}

if (o.denominator == 0 && ceil)

{

o.denominator = 1;

}

else if (o.denominator == 0)

{

std::cout << "Incorrect input\n";

exit(1);

}

delete[] numeratorChar;

delete[] denominatorChar;

delete[] wholePart;

}

fraction::fraction(fraction& o)

{

this->numerator = o.numerator;

this->denominator = o.denominator;

}

fraction fraction::operator+(fraction& o) const

{

fraction resultFraction;

if (this->denominator == o.getDenominator())

{

resultFraction.setNumerator(this->numerator + o.getNumerator());

resultFraction.setDenominator(this->denominator);

}

else

{

resultFraction.setNumerator((this->numerator \* o.getDenominator()) + (o.getNumerator() \* this->denominator));

resultFraction.setDenominator(this->denominator \* o.getDenominator());

}

return resultFraction;

}

fraction operator+(fraction& frac, double dbl)

{

fraction dblFraction(dbl);

fraction result = frac + dblFraction;

return result;

}

fraction operator+(double dbl, fraction& frac)

{

fraction dblFraction(dbl);

fraction result = dblFraction + frac;

return result;

}

fraction operator+(int i, fraction& frac)

{

fraction iFraction(i, 1);

fraction result = iFraction + frac;

return result;

}

fraction operator+(fraction& frac, int i)

{

fraction iFraction(i, 1);

fraction result = iFraction + frac;

return result;

}

fraction fraction::operator+=(fraction& frac)

{

\*this = \*this + frac;

return \*this;

}

fraction fraction::operator+=(double dbl)

{

fraction dblFraction(dbl);

\*this = \*this + dblFraction;

return \*this;

}

fraction fraction::operator+=(int i)

{

fraction iFraction(i, 1);

\*this = \*this + iFraction;

return \*this;

}

fraction fraction::operator=(fraction const& o)

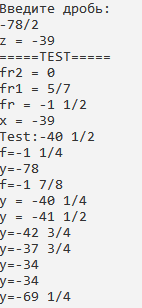
{

this->numerator = o.numerator;

this->denominator = o.denominator;

return \*this;

}

**Тестирование  
Изображение выглядит как текст, снимок экрана, Шрифт, рукописный текст

Автоматически созданное описаниеИзображение выглядит как текст, Шрифт, рукописный текст, снимок экрана

Автоматически созданное описаниеИзображение выглядит как текст, Шрифт, рукописный текст, снимок экрана

Автоматически созданное описаниеИзображение выглядит как текст, снимок экрана, Шрифт, число

Автоматически созданное описаниеИзображение выглядит как текст, снимок экрана, Шрифт, число

Автоматически созданное описаниеИзображение выглядит как текст, снимок экрана, Шрифт

Автоматически созданное описаниеИзображение выглядит как текст, снимок экрана, Шрифт, число

Автоматически созданное описаниеИзображение выглядит как текст, Шрифт, снимок экрана, рукописный текст

Автоматически созданное описание**