**Московский государственный технический**

**университет им. Н.Э. Баумана**

Факультет «Информатика и системы управления»

Кафедра ИУ5 «Системы обработки информации и управления»

Курс «Программирование на основе классов и шаблонов»

Отчет по лабораторной работе №3

«Классы. Перегрузка конструкторов и операций»

|  |  |  |
| --- | --- | --- |
| Выполнил: |  | Проверил: |
| студент группы ИУ5-23Б |  | преподаватель каф. ИУ5 |
| Лачина Екатерина |  | Аладин Д.В. |
| Подпись и дата: |  | Подпись и дата: |

Москва, 2024 г.

1. **Задача**

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.
3. **Листинг**

**Папка Laba33**

**CMakeLists.txt**

cmake\_minimum\_required(VERSION 3.2)

set(project "Laba33")

project(**${project}**)

set(CMAKE\_CXX\_STANDARD 17)

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

Laba33.cpp)

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

Laba33.h)

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

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

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

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

STATIC

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

**Laba33.cpp**

#include "Laba33.h"

Fraction::Fraction() {

numerator = 0;

denominator = 1;

}

Fraction::Fraction(const char\* input) {

int i = 0;

int j = 0;

int e = 0;

int cell\_part = 0;

// Проверка на то, есть ли целая часть в дроби

char maybe\_numerator[100];

if (is\_space(input)) {

while (input[e] != ' ') {

maybe\_numerator[i] = input[e];

e++;

i++;

}

maybe\_numerator[i] = '\0';

cell\_part = std::atoi(maybe\_numerator);

e++;

}

char numBuffer[100];

while (input[e] != '/' && input[e] != '\0') {

numBuffer[j] = input[e];

j++;

e++;

}

numBuffer[j] = '\0';

// Пропускаем разделитель и пробел

numerator = std::atoi(numBuffer);

if (input[e] != '\0') {

e++;

}

j = 0;

// Считываем знаменатель

char denBuffer[100];

if (input[e] != '\0') {

while (input[e] != '\0' && input[e] != ' ') {

denBuffer[j] = input[e];

e++;

j++;

}

denBuffer[j] = '\0';

denominator = std::atoi(denBuffer);

}

else {

denominator = 1;

}

//итог:

if (denominator != 0) {

if (cell\_part != 0) {

numerator = cell\_part \* (denominator \* abs(cell\_part) + numerator) / abs(cell\_part);

}

else {

if (is\_space(input) && input[0] == '-' && input[1] == '0') {

numerator = -1 \* numerator;

}

else {

numerator = (denominator \* abs(cell\_part) + numerator);

}

}

}

else {

std::cout << "знаменатель не может быть равен нулю!" << std::endl;

numerator = 0;

denominator = 1;

}

//если "-0 4/3" -4/3

}

Fraction::Fraction(char\* input) {

int i = 0;

int j = 0;

int e = 0;

int cell\_part = 0;

// Проверка на то, есть ли целая часть в дроби

char maybe\_numerator[100];

if (is\_space(input)) {

while (input[e] != ' ') {

maybe\_numerator[i] = input[e];

e++;

i++;

}

maybe\_numerator[i] = '\0';

cell\_part = std::atoi(maybe\_numerator);

e++;

}

char numBuffer[100];

while (input[e] != '/' && input[e] != '\0') {

numBuffer[j] = input[e];

//std::cout << numBuffer[j] << std::endl;

j++;

e++;

}

numBuffer[j] = '\0';

// Пропускаем разделитель и пробел

numerator = std::atoi(numBuffer);

if (input[e] != '\0') {

e++;

}

j = 0;

// Считываем знаменатель

char denBuffer[100];

if (input[e] != '\0') {

while (input[e] != '\0' && input[e] != ' ') {

denBuffer[j] = input[e];

e++;

j++;

}

denBuffer[j] = '\0';

denominator = std::atoi(denBuffer);

}

else {

denominator = 1;

}

//итог:

if (denominator != 0) {

if (cell\_part != 0) {

numerator = cell\_part \* (denominator \* abs(cell\_part) + numerator) / abs(cell\_part);

}

else {

if (is\_space(input) && input[0] == '-' && input[1] == '0') {

numerator = -1 \* numerator;

}

else {

numerator = (denominator \* abs(cell\_part) + numerator);

}

}

}

else {

std::cout << "знаменатель не может быть равен нулю!" << std::endl;

numerator = 0;

denominator = 1;

}

//если "-0 4/3" -4/3

}

Fraction::Fraction(double fraction) {

this->numerator = int(fraction) \* pow(10, N\_DEC) + int((fraction - int(fraction)) \* pow(10, N\_DEC));

this->denominator = pow(10, N\_DEC);

this->DeleteZeroes();

}

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

char input[100] = { 0 };

in.getline(input, 100, '\n');

Fraction temp(input);

fraction = temp;

return in;

}

std::ostream& operator<<(std::ostream& out, Fraction& obj) {

if (abs(obj.numerator) % obj.denominator == 0) {

out << obj.numerator / obj.denominator << '\n';

}

else {

bool isminus{ 1 };

if (obj.numerator < 0) { bool isminus = -1; }

int denominator = obj.denominator;

int integer = obj.numerator / obj.denominator;

int notint = abs(obj.numerator) % obj.denominator;

int n = std::min(notint, obj.denominator);

for (int i = 2; i < n; ++i) {

if (notint % i == 0 && denominator % i == 0) {

notint /= i;

denominator /= i;

--i;

}

}

if (integer == 0) { out << (int)((obj.numerator > 0) ? notint : -notint) << '/' << denominator << '\n'; }

else { out << integer << " " << notint << '/' << denominator << '\n'; };

}

return out;

}

Fraction operator+(const Fraction& a, const Fraction& b) {

return Fraction(a.numerator \* b.denominator + b.numerator \* a.denominator, a.denominator \* b.denominator);

}

Fraction& Fraction::operator=(const Fraction& b) {

numerator = b.numerator;

denominator = b.denominator;

return \*this;

}

Fraction operator+(const Fraction& a, double b) {

Fraction temp(b);

return a + temp;

}

Fraction operator+(double a, const Fraction& b) {

Fraction temp(a);

return temp + b;

}

Fraction operator+(const Fraction& a, int b) {

Fraction temp(b, 1);

return a + temp;

}

Fraction operator+(int a, const Fraction& b) {

Fraction temp(a, 1);

return temp + b;

}

Fraction& operator+=(Fraction& a, const Fraction& b) {

a = a + b;

return a;

}

Fraction& operator+=(Fraction& a, double b) {

a = a + b;

return a;

}

Fraction& operator+=(Fraction& a, int b) {

a = a + b;

return a;

}

**Laba33.h**

#pragma once

#define N\_DEC 4

#include <iostream>

#include <sstream>

#include <fstream>

#include <string>

#include <cstring>

class Fraction {

private:

int numerator;

int denominator;

bool is\_space(char\* input) {

for (int i = 0; input[i] != '\0'; i++) {

if (input[i] == ' ') {

return true;

}

}

return false;

}

bool is\_space(const char\* input) {

for (int i = 0; input[i] != '\0'; i++) {

if (input[i] == ' ') {

return true;

}

}

return false;

}

void DeleteZeroes() {

while (numerator % 10 == 0 && denominator % 10 == 0) {

numerator /= 10;

denominator /= 10;

}

}

public:

Fraction();

Fraction(int num, int den) : numerator(num), denominator(den) { }

Fraction(const char\* input);

Fraction(char\* input);

Fraction(double fraction);

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

friend std::ostream& operator<<(std::ostream& out, Fraction& obj);

friend Fraction operator+(const Fraction& a, const Fraction& b);

Fraction& operator=(const Fraction& b);

};

Fraction& operator+=(Fraction& a, int b);

Fraction& operator+=(Fraction& a, double b);

Fraction& operator+=(Fraction& a, const Fraction& b);

Fraction operator+(int a, const Fraction& b);

Fraction operator+(const Fraction& a, int b);

Fraction operator+(double a, const Fraction& b);

Fraction operator+(const Fraction& a, double b);

**Главные файлы**

**CMakeLists.txt**

cmake\_minimum\_required(VERSION 3.2)

set(project "Laba330")

project(**${project}**)

set(CMAKE\_CXX\_STANDARD 17)

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

main.cpp)

add\_subdirectory(Laba33)

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

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

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

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

target\_link\_libraries(**${project}** Laba33)

**main.cpp**

#include "Laba33/Laba33.h"

int main() {

setlocale(LC\_ALL, "Russian");

Fraction fraction;

//std::cin >> fraction;

//fraction.printFraction();

//std::cout << "answer: " << fraction << std::endl;

// ввод дроби с клавиатуры

std::cout << "Введите дробь: \n";

Fraction z;

std::cin >> z;

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

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;

double dbl0 = dbl / 2;

f += dbl0;

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;}

1. **Тестирование**

 

 

 