**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 |
| Поляков Данила |  | Правдина А.Д. |
| Подпись и дата: 25.02.2022 |  | Подпись и дата: 25.02.2022 |

Москва, 2022 г.

**Постановка задачи**

Создать класс для работы с обыкновенными дробями. Все операции, которые должны выполняться с дробями, включены в программу в приложении 1.

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).

Значения представленных выше дробей на экране при выводе должны иметь вид:

-2 1/3, 8, 6.

3/4, -3, -1 1/3.

При выводе и после выполнения арифметических операций дроби сокращаются, то есть числитель и знаменатель не должны иметь общих множителей.

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.

Для инициализации объектов разрабатываемого класса обыкновенных дробей предусмотреть соответствующие конструкторы (с одним аргументом типа char\*, с одним аргументом типа double и с двумя аргументами типа int, которые имеют значения по умолчанию).

При перегрузке операций использовать функции - члены класса, а где это невозможно, то функции - друзья класса.

Для обеспечения более удобного контроля результатов выполнения программы вставьте в конструкторы и перегруженные операции операторы вывода, идентифицирующие выполняемую функцию.

Выполните следующий эксперимент: закомментируйте операции дроби с ***int*** и повторно выполните программу. Объясните результаты сложения дробей с целыми числами.

**Текст программы**

Листинг кода программы:

**Fraction.hpp**

*// Copyright 2021 qqq <polyakovdd@student.bmstu.ru>  
  
  
#ifndef* CODE\_FRACTION\_HPP  
*#define* CODE\_FRACTION\_HPP  
  
*#define* N\_DEC 4  
*#define* MAX\_LENGTH 18  
  
*#include* "includes.hpp"  
  
*class* Fraction {  
 *bool* Sign;  
 *int* Integer;  
 *int* Numerator;  
 *int* Denominator;  
  
 *void* Normalize();  
  
  
 *int* getNod(*int*, *int*);  
  
*public*:  
  
  
 Fraction();  
  
 Fraction(*int*, *int*, *int*);  
  
 Fraction(*int*, *int*);  
 Fraction(*int*);  
  
 Fraction(*double*);  
  
 *double* ToDouble();  
  
 Fraction &*operator*=(*const char* \*frStr) {  
 *int* tempFrist, tempSecond, tempThird;  
 std::stringstream istr(frStr);  
 istr >> tempFrist;  
 *if* (istr.peek() == '/') {  
 istr.get();  
 istr >> tempSecond;  
 \**this* = Fraction(tempFrist, tempSecond);  
 } *else* {  
 istr >> tempSecond;  
 istr.get();  
 istr >> tempThird;  
 \**this* = Fraction(tempFrist, tempSecond, tempThird);  
 }  
 *this*->Normalize();  
 *return* \**this*;  
 }  
  
 Fraction &*operator*=(*const* Fraction &fr) {  
 Sign = fr.Sign;  
 Integer = fr.Integer;  
 Numerator = fr.Numerator;  
 Denominator = fr.Denominator;  
  
 *this*->Normalize();  
 *return* \**this*;  
 }  
  
 Fraction &*operator*=(*const double* &fr) {  
 *double* q = fr;  
 *if* (q < 0) {  
 Sign = 1;  
 } *else* Sign = 0;  
 q = abs(q);  
 Integer = *static\_cast*<*int*>(q);  
 q -= Integer;  
 Numerator = *static\_cast*<*int*>(q \* pow(10, N\_DEC));  
 Denominator = *static\_cast*<*int*>(pow(10, N\_DEC));  
  
 *this*->Normalize();  
 *return* \**this*;  
 }  
  
 *friend* Fraction &*operator*+=(Fraction &left, *const* Fraction &right);  
  
 *friend* std::istream &*operator*>>(std::istream &, Fraction &);  
  
 *friend* std::ostream &*operator*<<(std::ostream &, Fraction &);  
  
 *friend* Fraction *operator*+(*const* Fraction &, *const* Fraction &);  
  
 *friend* Fraction *operator*+(*const* Fraction &, *const int* &); *//как тут лучше то, чтобы не плодить перегрузки  
 friend* Fraction *operator*+(*const int* &, *const* Fraction &);  
  
 *friend* Fraction *operator*+(*const* Fraction &, *const double* &);  
  
 *friend* Fraction *operator*+(*const double* &,*const* Fraction &);  
};  
  
  
*#endif //CODE\_FRACTION\_HPP*

**Fraction.cpp**

*// Copyright 2021 qqq <polyakovdd@student.bmstu.ru>  
  
  
#include* "Fraction.hpp"  
  
  
Fraction::Fraction() {  
 Integer = 0;  
 Numerator = 0;  
 Denominator = 0;  
}  
  
Fraction::Fraction(*int* intgr, *int* num, *int* den) {  
 *if* (intgr < 0) {  
 Sign = 1;  
 } *else* Sign = 0;  
 Integer = abs(intgr);  
 Numerator = num;  
 Denominator = den;  
 *this*->Normalize();  
}  
  
Fraction::Fraction(*int* num, *int* den) {  
 Integer = 0;  
 *if* (num < 0) {  
 Sign = 1;  
 } *else* Sign = 0;  
 Numerator = abs(num);  
 Denominator = den;  
 *this*->Normalize();  
}  
Fraction::Fraction(*int* q){  
 *if* (q<0)  
 {  
 Sign = *true*;  
 }  
 *else* Sign = *false*;  
 Integer = q;  
 Numerator = 0;  
 Denominator = 1;  
}  
Fraction::Fraction(*const double* fr) {  
 *double* q = fr;  
 *if* (fr < 0) {  
 Sign = 1;  
 } *else* Sign = 0;  
 q = abs(fr);  
 Integer = *static\_cast*<*int*>(q);  
 q -= Integer;  
 Numerator = *static\_cast*<*int*>(q \* pow(10, N\_DEC));  
 Denominator = *static\_cast*<*int*>(pow(10, N\_DEC));  
  
 *this*->Normalize();  
}  
  
*int* Fraction::getNod(*int* a, *int* b) {  
 *if* (b == 0)  
 *return* a;  
 *return* getNod(b, a % b);  
}  
  
*void* Fraction::Normalize() {  
 Integer += Numerator / Denominator;  
 Numerator = Numerator % Denominator;  
  
 *int* nod = getNod(Numerator, Denominator);  
 Denominator /= nod;  
 Numerator /= nod;  
}  
  
std::istream &*operator*>>(std::istream &istr, Fraction &fraction) {  
 *int* tempFirst, tempSecond, tempThird;  
 istr >> tempFirst;  
 *if* (istr.peek() == '/') {  
 istr.get();  
 istr >> tempSecond;  
 fraction = Fraction(tempFirst, tempSecond);  
 } *else if* (istr.peek() == '\n') {  
 fraction = Fraction(tempFirst);  
 } *else* {  
 istr >> tempSecond;  
 istr.get();  
 istr >> tempThird;  
 fraction = Fraction(tempFirst, tempSecond, tempThird);  
 fraction.Normalize();  
 }  
 *return* istr;  
}  
  
std::ostream &*operator*<<(std::ostream &ostr, Fraction &fraction) {  
 *if* (fraction.Numerator != 0) {  
 ostr << '\n';  
 ostr << " " << fraction.Numerator << "\n ";  
 *if* (fraction.Sign) ostr << '-';  
 *if* (fraction.Integer != 0) ostr << fraction.Integer;  
 ostr << " ---\n";  
 ostr << " " << fraction.Denominator << '\n';  
 *return* ostr;  
 } *else* {  
 *if* (fraction.Sign) ostr << '-';  
 ostr << fraction.Integer;  
 *return* ostr;  
 }  
  
}  
  
Fraction *operator*+(*const* Fraction &fraction1, *const* Fraction &fraction2) {  
 Fraction fr1 = fraction1;  
 Fraction fr2 = fraction2;  
 *double* q1 = fr1.ToDouble();  
 *double* q2 = fr2.ToDouble();  
 q1 = q1 + q2;  
 Fraction result = q1;  
 *return* result;  
}  
  
Fraction *operator*+(*const* Fraction &fraction1, *const double* &dbl) {  
 Fraction fr = fraction1;  
 *double* q1 = fr.ToDouble();  
 q1 = q1 + dbl;  
 Fraction result = q1;  
 *return* result;  
}  
  
Fraction *operator*+(*const double* &dbl, *const* Fraction &fraction1) {  
 Fraction fr = fraction1;  
 *double* q1 = fr.ToDouble();  
 q1 = q1 + dbl;  
 Fraction result = q1;  
 *return* result;  
}  
  
Fraction *operator*+(*const* Fraction &fraction1, *const int* &intgr) {  
 Fraction fr = fraction1;  
 *if* (fr.Sign) fr.Integer = fr.Integer \* (-1) + intgr;  
 *else* fr.Integer += intgr;  
 *if* (fr.Integer < 0) {  
 fr.Sign = *true*;  
 } *else* fr.Sign = *false*;  
 fr.Normalize();  
 *return* fr;  
}  
  
Fraction *operator*+(*const int* &intgr, *const* Fraction &fraction1) {  
 Fraction fr = fraction1;  
 *if* (fr.Sign) fr.Integer = fr.Integer \* (-1) + intgr;  
 *else* fr.Integer += intgr;  
 *if* (fr.Integer < 0) {  
 fr.Sign = *true*;  
 } *else* fr.Sign = *false*;  
 fr.Normalize();  
 *return* fr;  
}  
  
Fraction &*operator*+=(Fraction &left, *const* Fraction &right) {  
 *//left = left+right;//бред с константной перегрузкой сначала просто суммы, а потом ломающимся toDouble* Fraction q = right;  
 left = left + q;  
 *return* left;  
}  
  
*double* Fraction::ToDouble() {  
 *double* qwe = *static\_cast*<*double*>(Numerator) / Denominator;  
 *double* q = *static\_cast*<*double*>(*this*->Integer) + qwe;  
 *if* (Sign) q \*= (-1);  
 *return* q;  
}

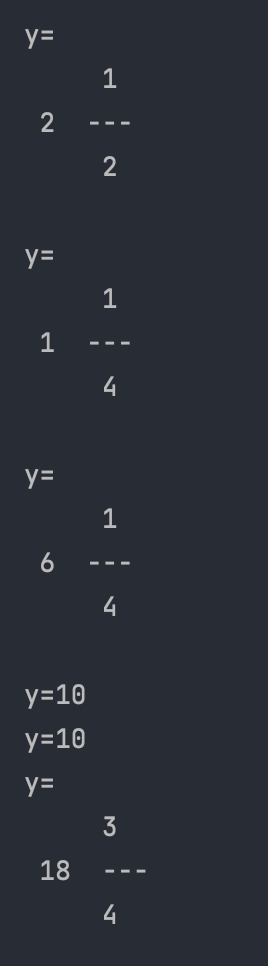
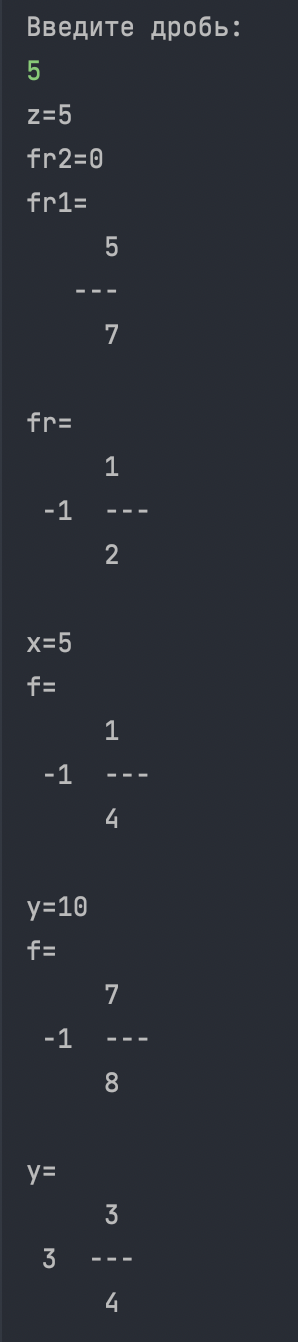
**includes.hpp**

*// Copyright 2021 qqq <polyakovdd@student.bmstu.ru>  
  
  
#ifndef* CODE\_INCLUDES\_HPP  
*#define* CODE\_INCLUDES\_HPP  
  
  
*#include* <iostream>  
*#include* <iomanip>  
*#include* <cmath>  
*#include* <algorithm>  
*#include* <sstream>  
  
*#endif //CODE\_INCLUDES\_HPP*

**main.cpp**

*#include* "includes.hpp"  
*#include* "Fraction.hpp"  
  
*int* main() {  
*// Fraction q;  
// q = "10/4";  
// std::std::cout << "Fraction: " << q << '\n';  
// q = "-5 1/2";  
// std::std::cout << q << '\n';  
// q = -5.125;  
// std::std::cout << q << '\n';  
// q = Fraction(10.256);  
// std::std::cout << q << '\n';  
// Fraction z;  
// std::std::cout << "Enter Fraction\n";  
// std::std::cin >> z;  
// Fraction qz = z+q;  
// std::std::cout << qz;* 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;  
 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;  
 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;  
}

**Анализ результатов**

****

