

**Московский государственный технический
университет им. Н.Э. Баумана.**

Факультет «Информатика и системы управления»

Кафедра ИУ5. Курс «Программирование на основе классов и шаблонов»

Отчет по лабораторной работе № 2

«Классы. Перегрузка конструкторов и операций»

Выполнил:

студент группы ИУ5-23

Терентьев Владислав

Подпись и дата:

Проверил:

преподаватель каф. ИУ5

Козлов А. Д.

Подпись и дата:

Москва, 2019 г.

1. Постановка задачи

Создать класс "fraction" для работы с обыкновенными дробями. Реализовать выполнение арифметических операций для объектов этого класса и перегрузить соответствующие операторы. Реализовать конструкторы с параметрами типа: число (числитель); 2 числа (числитель, знаменатель); 3 числа (целая часть, числитель, знаменатель), а так же перегрузить операторы ввода и вывода.

2. Разработка интерфейса класса

В программе описан класс **fraction** с переменными-членами (спецификатор доступа **private**): sign типа **short** для хранения знака дроби; numerator типа **long int** для хранения числителя дроби; denominator типа **long int** для хранения знаменателя дроби. Класс содержит (спецификатор доступа **public**): **getters** для переменных-членов класса; **конструкторы** (default, копирующий, с параметрами типа: число **int**; число **double**; числа **int** и **int**; числа **double** и **int**; числа **int**, **int** и **int**; числа **int**, **double** и **int**); **операторы** **=**, **+**, *****, **-**, **/**, **++**, **--**, **+=**, **-=** типа **fraction** и с параметром из набора типов **int**, **double**, **fraction**; **операторы** **double**, **int**, **long int**; **операторы** **-**, **+**, **++**, **--** типа **fraction** без параметров.

Программа содержит (вне класса) перегруженные **операторы**: **+**, **-**, *****, **/** типа **fraction** с двумя параметрами (первый – типа **int** или **double**, второй – типа **fraction**); **<<** типа **std::ostream&** и параметрами типа (**std::ostream &**, **fraction**); **>>** типа **std::istream&** и параметрами типа (**std::istream &**, **fraction &**); **==**, **!=**, **>**, **<**, **<=**, **>=** типа **bool** и параметрами типа (**fraction &**, **fraction &**). Так же содержит **функцию** **Nod** типа **long int** с параметрами типа (**long int**, **long int**) для нахождения наибольшего общего делителя.

Для перегрузки арифметических и логических операторов с дробями достаточно разработать только операции **+**, *****, **/**, **==** (например: операция $d1 - d2$ это $d1 + (-1) * d2$; операция $d1 != d2$ это $\neg(d1 == d2)$; операция $d1 > d2$ это сравнение знака полученной дроби из результата операции $d1 - d2$ с нулем; операция $d1 <= d2$ это $\neg(d1 > d2)$ и т.д.).

3. Разработка алгоритма

Схема алгоритма функции Nod:

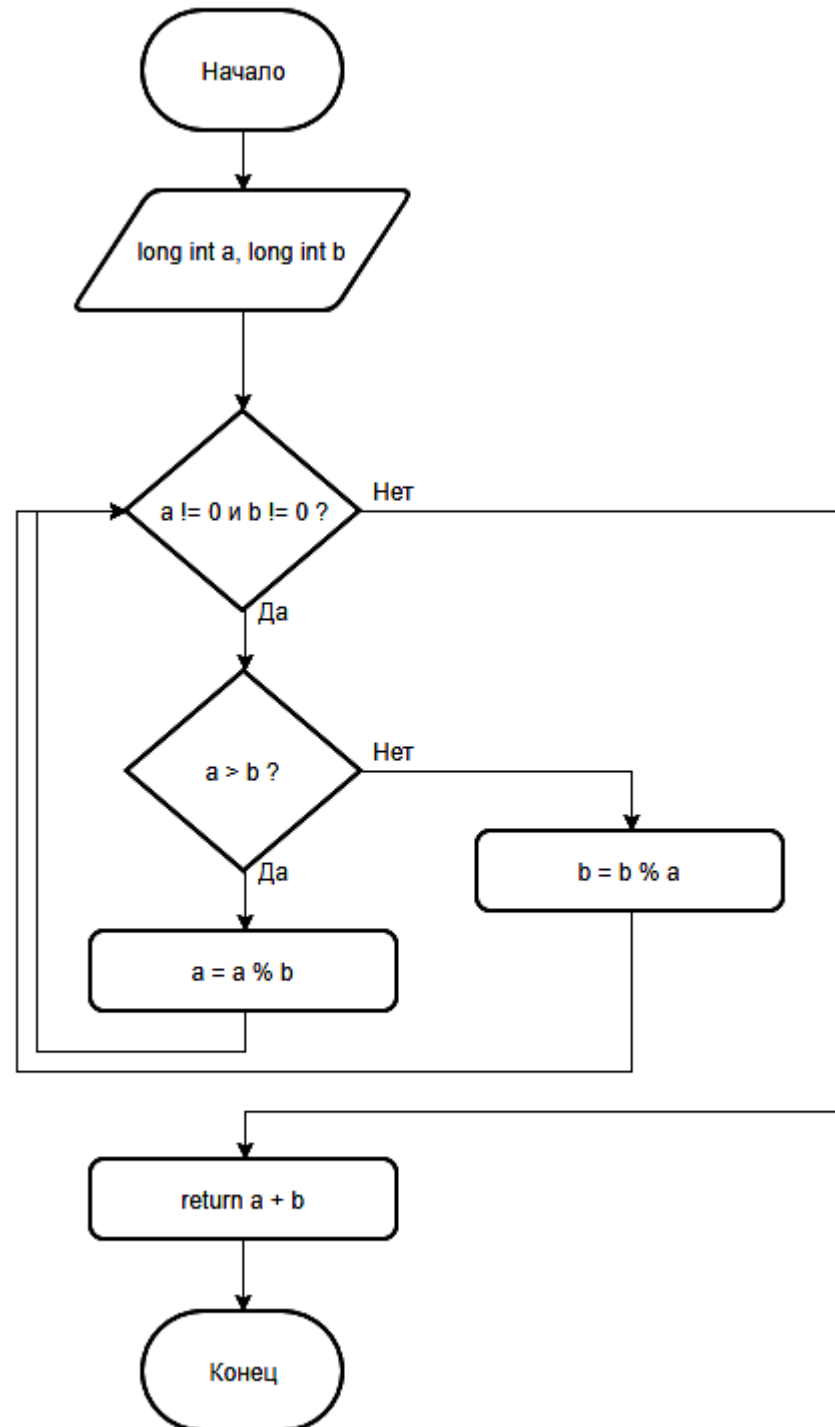
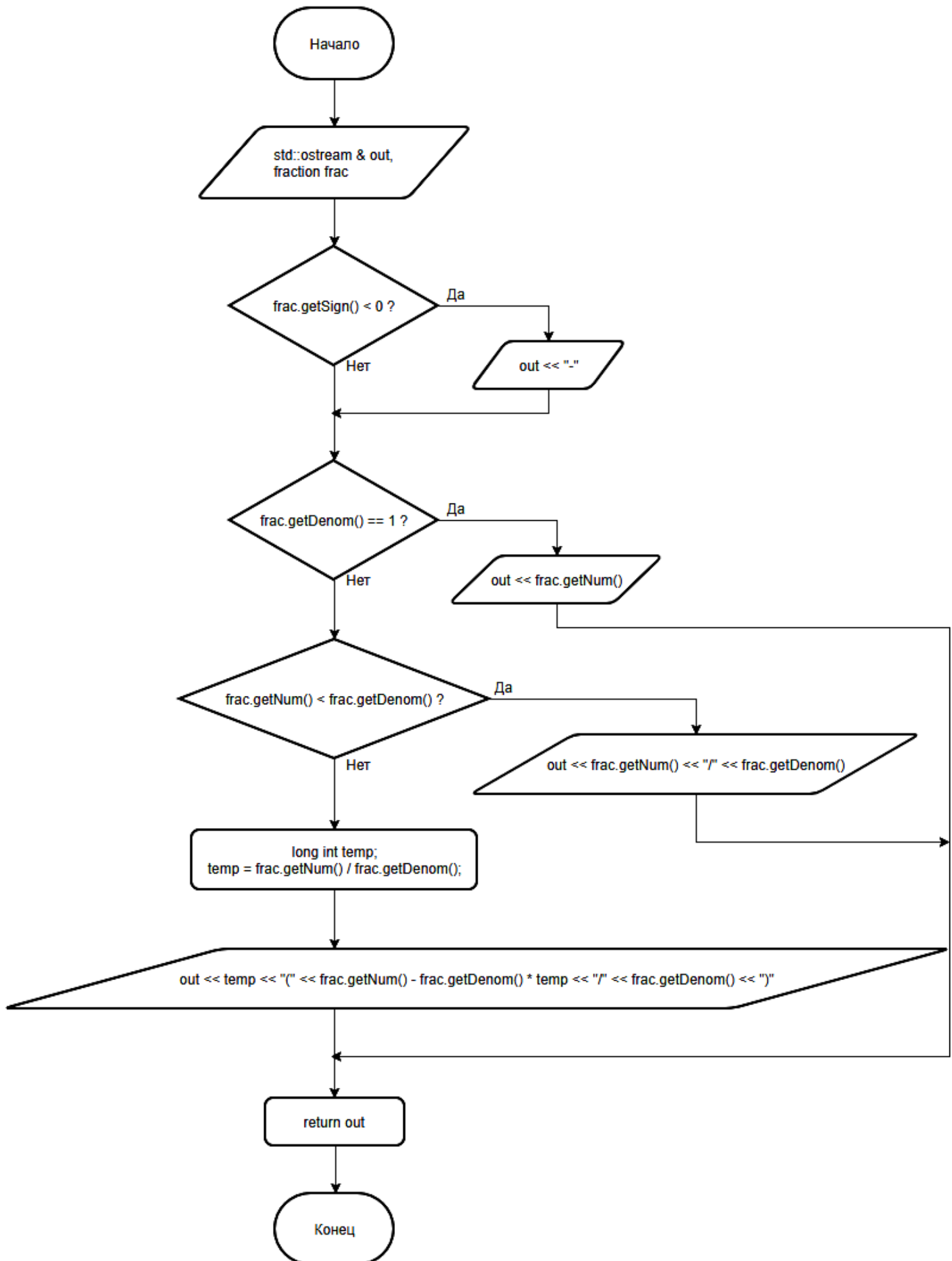


Схема алгоритма оператора вывода для дроби:



4. Текст программы

```
#include <iostream>
#include <cassert>

using namespace std;

long int Nod(long int a, long int b)
{
    while (a != 0 && b != 0)
        if (a > b) {
            a = a % b;
        }
        else {
            b = b % a;
        }
    return a + b;
}

class fraction {
private:
    short sign;

    long int numerator;

    long int denominator;

public:
    short getSign() {
        return sign;
    }

    long int getNum() {
        return numerator;
    }

    long int getDenom() {
        return denominator;
    }

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

    fraction(const fraction& frac) :sign(frac.sign), numerator(frac.numerator),
denominator(frac.denominator) {}

    fraction(int currentNum) {
        if (currentNum >= 0) {
            sign = 1;
        }
        else {
            sign = -1;
            currentNum = -1 * currentNum;
        }
        numerator = int(currentNum);
        denominator = 1;
    }

    fraction(double currentNum) {
        long int nod;
        if (currentNum >= 0) {
            sign = 1;
        }
        else {
            sign = -1;
            currentNum = -1 * currentNum;
        }
        numerator = long int(currentNum * 100000);
    }
}
```

```

        denominator = 100000;
        nod = Nod(numerator, denominator);
        numerator = numerator / nod;
        denominator = denominator / nod;
    }

fraction(int currentNum, int currentDenom) {
    assert(currentDenom != 0);
    long int nod;
    if (currentNum >= 0) {
        sign = 1;
    }
    else {
        sign = -1;
        currentNum = -1 * currentNum;
    }
    if (currentDenom < 0) {
        sign = -1 * sign;
        currentDenom = -1 * currentDenom;
    }
    nod = Nod(currentNum, currentDenom);
    numerator = int(currentNum) / nod;
    denominator = int(currentDenom) / nod;
}

fraction(double currentNum, int currentDenom) {
    assert(currentDenom != 0);
    long int nod;
    if (currentNum >= 0) {
        sign = 1;
    }
    else {
        sign = -1;
        currentNum = -1 * currentNum;
    }
    if (currentDenom < 0) {
        sign = -1 * sign;
        currentDenom = -1 * currentDenom;
    }
    numerator = long int(currentNum * 100000);
    denominator = 100000 * int(currentDenom);
    nod = Nod(numerator, denominator);
    numerator = numerator / nod;
    denominator = denominator / nod;
}

fraction(int ent, int currentNum, int currentDenom) {
    if (ent >= 0) {
        *this = fraction(currentNum + ent * currentDenom, currentDenom);
    }
    else {
        ent = -1 * ent;
        *this = fraction(-(currentNum + ent * currentDenom), currentDenom);
    }
}

fraction(int ent, double currentNum, int currentDenom) {
    if (ent >= 0) {
        *this = fraction(currentNum + ent * currentDenom, currentDenom);
    }
    else {
        ent = -1 * ent;
        *this = fraction(-(currentNum + ent * currentDenom), currentDenom);
    }
}

fraction operator-() {
    sign = -1 * sign;
    return *this;
}

```

```

}

fraction operator+() {
    return *this;
}

explicit operator double() {
    return double(numerator) / denominator;
}

explicit operator int() {
    return int(numerator / denominator);
}

explicit operator long int() {
    return numerator / denominator;
}

fraction& operator= (const fraction & frac) {
    if (this == &frac) {
        return *this;
    }
    sign = frac.sign;
    numerator = frac.numerator;
    denominator = frac.denominator;
    return *this;
}

fraction operator+ (fraction frac) {
    return fraction(this->sign * this->numerator * frac.denominator + frac.sign *
frac.numerator * this->denominator, this->denominator * frac.denominator);
}

fraction operator+ (int val) {
    return fraction(this->sign * this->numerator + val * this->denominator, this-
>denominator);
}

fraction operator+ (double val) {
    return fraction(this->sign * this->numerator + val * this->denominator, this-
>denominator);
}

fraction operator* (fraction & frac) {
    return fraction(this->sign * frac.sign * this->numerator * frac.numerator, this-
>denominator * frac.denominator);
}

fraction operator* (int val) {
    return fraction(this->sign * this->numerator * val, this->denominator);
}

fraction operator* (double val) {
    return fraction(this->sign * this->numerator * val, this->denominator);
}

fraction operator- (fraction frac) {
    return *this + (-frac);
}

fraction operator- (int val) {
    return *this + (-1 * val);
}

fraction operator- (double val) {
    return *this + (-1 * val);
}

fraction operator/ (fraction & frac) {

```

```

        return fraction(this->sign * frac.sign * this->numerator * frac.denominator, this->denominator * frac.numerator);
    }

    fraction operator/ (int val) {
        return fraction(this->sign * this->numerator, this->denominator * val);
    }

    fraction operator/ (double val) {
        return fraction(this->sign * this->numerator, this->denominator * val);
    }

    fraction operator++ () {
        if ((sign < 0) && (numerator < denominator)) {
            sign = 1;
            numerator = denominator - numerator;
        }
        else {
            numerator = numerator + denominator;
        }
        return *this;
    }

    fraction operator-- () {
        if ((sign > 0) && (numerator < denominator)) {
            sign = -1;
            numerator = denominator - numerator;
        }
        else {
            numerator = numerator + denominator;
        }
        return *this;
    }

    fraction operator++ (int) {
        fraction temp = *this;
        ++(*this);
        return temp;
    }

    fraction operator-- (int) {
        fraction temp = *this;
        --(*this);
        return temp;
    }

    fraction operator+= (fraction frac) {
        *this = *this + frac;
        return *this;
    }

    fraction operator+= (long int val) {
        *this = *this + val;
        return *this;
    }

    fraction operator+= (int val) {
        *this = *this + val;
        return *this;
    }

    fraction operator+= (double val) {
        *this = *this + val;
        return *this;
    }

    fraction operator-= (fraction frac) {
        *this = *this - frac;
        return *this;
    }

```



```

    }

    fraction operator-= (long int val) {
        *this = *this - val;
        return *this;
    }

    fraction operator-= (int val) {
        *this = *this - val;
        return *this;
    }

    fraction operator-= (double val) {
        *this = *this - val;
        return *this;
    }
};

fraction operator+ (int val, fraction frac) {
    return fraction(val * frac.getDenom() + frac.getSign() * frac.getNum(), frac.getDenom());
}

fraction operator+ (double val, fraction & frac) {
    return fraction(val * frac.getDenom() + frac.getSign() * frac.getNum(), frac.getDenom());
}

fraction operator- (int val, fraction frac) {
    return val + (-frac);
}

fraction operator- (double val, fraction frac) {
    return val + (-frac);
}

fraction operator* (int val, fraction & frac) {
    return fraction(frac.getSign() * frac.getNum() * val, frac.getDenom());
}

fraction operator* (double val, fraction & frac) {
    return fraction(frac.getSign() * frac.getNum() * val, frac.getDenom());
}

fraction operator/ (int val, fraction & frac) {
    return fraction(frac.getNum(), frac.getSign() * frac.getDenom() * val);
}

fraction operator/ (double val, fraction & frac) {
    return fraction(frac.getNum(), frac.getSign() * frac.getDenom() * val);
}

std::ostream& operator<< (std::ostream & out, fraction frac) {
    if (frac.getSign() < 0) {
        out << "-";
    }
    if (frac.getDenom() == 1) {
        out << frac.getNum();
    }
    else {
        if (frac.getNum() < frac.getDenom()) {
            out << frac.getNum() << "/" << frac.getDenom();
        }
        else {
            long int temp;
            temp = frac.getNum() / frac.getDenom();
            out << temp << "(" << frac.getNum() - frac.getDenom() * temp << "/" <<
frac.getDenom() << ")";
        }
    }
    return out;
}

```

```

}

std::istream& operator>> (std::istream & in, fraction & frac)
{
    double currentNum = 0, temp;
    long int currentDenom = 1;
    int sym;
    in >> temp;
    sym = in.get();
    if (sym == 10) {
        frac = fraction(temp);
    }
    else {
        if (sym == 47) {
            in >> currentDenom;
            frac = fraction(temp, currentDenom);
        }
        else {
            if (sym == 40) {
                in >> currentNum;
                in.get();
                in >> currentDenom;
                in.get();
                frac = fraction(int(temp), currentNum, currentDenom);
            }
        }
    }
    return in;
}

bool operator== (fraction & frac1, fraction & frac2) {
    return ((frac1.getSign() == frac2.getSign()) && (frac1.getDenom() == frac2.getDenom()) &&
    (frac1.getNum() == frac2.getNum()));
}

bool operator!= (fraction & frac1, fraction & frac2) {
    return !(frac1 == frac2);
}

bool operator> (fraction & frac1, fraction & frac2) {
    fraction temp(frac1 - frac2);
    if (temp.getSign() > 0) {
        return 1;
    }
    else {
        return 0;
    }
}

bool operator< (fraction & frac1, fraction & frac2) {
    fraction temp(frac1 - frac2);
    if (temp.getSign() < 0) {
        return 1;
    }
    else {
        return 0;
    }
}

bool operator<= (fraction & frac1, fraction & frac2) {
    return !(frac1 > frac2);
}

bool operator>= (fraction & frac1, fraction & frac2) {
    return !(frac1 < frac2);
}

void main()
{

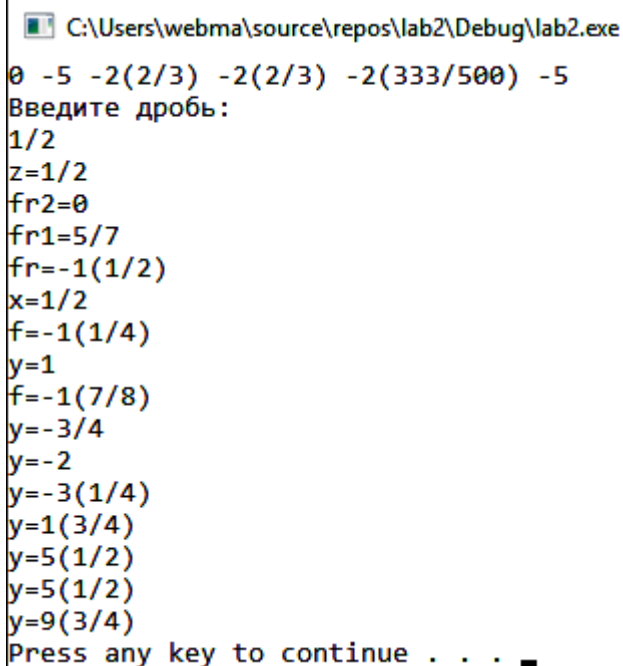
```

```

setlocale(LC_ALL, "Russian");
fraction a, b(-5), c(-8, 3), d(-2, 2, 3), e(-2.666), f_1(b);
cout << a << " " << b << " " << c << " " << d << " " << e << " " << f_1 << endl;
//ввод дроби с клавиатуры
cout << "Введите дробь: \n";
fraction z;
cin >> z;
cout << "z=" << z << endl;
//проверка конструкторов
fraction fr1(10, 14), fr2;
cout << "fr2=" << fr2 << endl;
cout << "fr1=" << fr1 << endl;
fraction fr(-1, 4, 8);
cout << "fr=" << fr << endl;
fraction x(z), y;
cout << "x=" << x << endl;
double dbl = -1.25;
fraction f = dbl;
cout << "f=" << f << endl;
//проверка перегруженной операции "+"
y = x + z;
cout << "y=" << y << endl;
y += x;
f += dbl / 2;
cout << "f=" << f << endl;
y = x + dbl;
cout << "y=" << y << endl;
y = dbl + y;
cout << "y=" << y << endl;
y += dbl;
cout << "y=" << y << endl;
int i = 5;
y += i;
cout << "y=" << y << endl;
y = i + x;
cout << "y=" << y << endl;
y = x + i;
cout << "y=" << y << endl;
y += dbl + i + x;
cout << "y=" << y << endl;
system("pause");
}

```

5. Анализ результатов



```

C:\Users\webma\source\repos\lab2\Debug\lab2.exe
0 -5 -2(2/3) -2(2/3) -2(333/500) -5
Введите дробь:
1/2
z=1/2
fr2=0
fr1=5/7
fr=-1(1/2)
x=1/2
f=-1(1/4)
y=1
f=-1(7/8)
y=-3/4
y=-2
y=-3(1/4)
y=1(3/4)
y=5(1/2)
y=5(1/2)
y=9(3/4)
Press any key to continue . . .

```

Программа работает исправно.