Задание.

Цель работы: научится перегружать операции в классе.

Задания

В каждом упражнении требуется реализовать в том или ином виде определение нового класса. Во всех заданиях необходимо реализовать конструктор инициализации (один или несколько) и конструктор без аргументов.

Для демонстрации работы с объектами нового типа во всех заданиях требуется написать главную функцию. В программе обязательно должны быть продемонстрированы различные способы создания объектов и массивов объектов. Программа должна демонстрировать использование всех функций и методов.

15. Создать класс Fraction для работы с дробными числами. Число должно быть представлено двумя полями: целая часть - длинное целое со знаком, дробная часть - беззнаковое короткое целое. Реализовать арифметические операции сложения, вычитания, умножения и операции сравнения.

Код.

```
#include <iostream>
#include <cmath>
#include <string>

using namespace std;

class Fraction
{
   long int whole_part; // более конкретно - числитель.
   unsigned short int fractional_part; // более конкретно -
знаменатель.

   void fractional_cast(Fraction &a); // приведение дробей к общему
знаменателю.
   void fractional_reduction(); // сокращение дроби при помощи
алгоритма Евклида.

   public:
       Fraction(); // конструктор без параметров.
       Fraction(int wp, int fp); // "стандартный" конструктор.
       Fraction(const Fraction &a); // конструктор копирования.

       friend istream& operator>>(istream& stream, Fraction &a);
       friend ostream& operator<<(ostream& stream, const Fraction &a);
```

```
        friend Fraction operator+(Fraction &a, Fraction &b);
        friend Fraction operator-(Fraction &a, Fraction &b);
        friend Fraction operator*(Fraction &a, Fraction &b);
        friend bool operator==(Fraction &a, Fraction &b);
        friend bool operator>(Fraction &a, Fraction &b);
        friend bool operator<(Fraction &a, Fraction &b);
};

Fraction::Fraction()
{
    this->whole_part = 1;
    this->fractional_part = 1;
}

Fraction::Fraction(int wp, int fp)
{
    if (fp > 0){ this->whole_part = wp; this->fractional_part = fp;
this->fractional_reduction(); return; }
    this->whole_part = 1;
    this->fractional_part = 1;
    cout << "Object was not created correctly(were setted default
values)! You've entered a wrong data!\n";
}

Fraction::Fraction(const Fraction &a)
{
    this->whole_part = a.whole_part;
    this->fractional_part = a.fractional_part;
}

void Fraction::fractional_cast(Fraction &a)
{
    this->whole_part *= a.fractional_part;
    a.whole_part *= this->fractional_part;
    this->fractional_part *= a.fractional_part;
    a.fractional_part = this->fractional_part;
}

void Fraction::fractional_reduction()
{
    int x = abs(this->whole_part), y = this->fractional_part;
    while (x != 0 && y != 0)
```

```
    {
        (x > y) ? x %= y : y %= x;
    }
    // (x + y) - наибольший общий делитель.
    this->whole_part /= (x + y);
    this->fractional_part /= (x + y);
}

istream& operator>>(istream& stream, Fraction &a)
{
    int wp, fp;
    cout << "Enter a whole part: ";
    stream >> wp;
    cout << "Enter a fractional part: ";
    stream >> fp;
    if (fp > 0){ a.whole_part = wp; a.fractional_part = fp;
a.fractional_reduction(); return stream; }
    cout << "Incorrect values!\n"; return stream;
}

ostream& operator<<(ostream& stream, const Fraction &a)
{
    if (a.whole_part == 0){ return (stream << 0); }
    else if (a.whole_part == 1 && a.fractional_part == 1){ return
(stream << 1); }
    return (stream << a.whole_part << "/" << a.fractional_part);
}

Fraction operator+(Fraction &a, Fraction &b)
{
    Fraction x(a), y(b);
    x.fractional_cast(y);
    x.whole_part += y.whole_part;
    x.fractional_reduction();
    return x;
}

// Спорный момент с нулём есть - если выходим в нуль, то нуль только
числитель, не знаменатель. Учёл этот момент в выводе(то же сделал и для
единицы).
Fraction operator-(Fraction &a, Fraction &b)
{
    Fraction x(a), y(b);
```

```
    x.fractional_cast(y);
    x.whole_part -= y.whole_part;
    x.fractional_reduction();
    return x;
}

Fraction operator*(Fraction &a, Fraction &b)
{
    Fraction x(a), y(b);
    x.whole_part *= y.whole_part;
    x.fractional_part *= y.fractional_part;
    x.fractional_reduction();
    return x;
}

bool operator==(Fraction &a, Fraction &b)
{
    return (a.whole_part == b.whole_part && a.fractional_part ==
b.fractional_part);
}

bool operator>(Fraction &a, Fraction &b)
{
    int x = a.whole_part * b.fractional_part, y = b.whole_part *
a.fractional_part;
    return (x > y);
}

bool operator<(Fraction &a, Fraction &b)
{
    int x = a.whole_part * b.fractional_part, y = b.whole_part *
a.fractional_part;
    return (x < y);
}

int main()
{
    Fraction tests[2];
    for (int i = 0; i < 2; i++)
    {
        cin >> tests[i];
    }
    Fraction test(tests[0]);
```

```
    cout << (test == tests[0]) << endl;
    tests[0] = tests[0] + tests[1];
    cout << tests[0] << endl;
    test = test * tests[0];
    cout << test << endl;
    test = test - tests[1];
    cout << test << endl;
    return 0;
}
```

Результат.

```
Enter a whole part: 1
Enter a fractional part: 2
Enter a whole part: 1
Enter a fractional part: 4
1
3/4
3/8
1/8
```