Задание.

Задания

Во всех заданиях, помимо указанных в задании операций, обязательно должны быть реализованы следующие методы:

– метод инициализации init;
– ввод с клавиатуры read;
– вывод на экран display;
– преобразование в строку toString.

Все задания должны быть реализованы двумя способами:

- тип данных представляется структурой с необходимыми полями, а операции реализуются как внешние функции, которые получают объекты данного типа в качестве аргументов;
– как класс с закрытыми полями, где операции реализуются как методы класса.

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>

using namespace std;

struct Fraction
{
   long int whole_part;
   unsigned short int fraction_part;
};

// Приведение дробей к общему знаменателю.
void fractional_cast(Fraction &a, Fraction &b)
{
   a.whole_part *= b.fraction_part;
   b.whole_part *= a.fraction_part;
   a.fraction_part *= b.fraction_part;
   b.fraction_part = a.fraction_part;
}

// Сокращение дробей при помощи алгоритма Евклида.
void fraction_reduction(Fraction &a)
{
   int x = abs(a.whole_part), y = a.fraction_part;
   while (x != 0 && y != 0)
```

```
    {
        (x > y) ? x%= y : y%=x;
    }
    a.whole_part /= (x + y);
    a.fraction_part /= (x + y);
}

void init(Fraction &a, int wp, int fp)
{
    if (fp > 0)
    {
        a.whole_part = wp;
        a.fraction_part = fp;
        return;
    }
    cout << "Incorrect data! Default values were setted!\n";
    a.whole_part = 1;
    a.fraction_part = 1;
}

void display(Fraction &a)
{
    cout << a.whole_part << "/" << a.fraction_part <<  "\n";
}

void read(Fraction &a)
{
    int wp, fp;
    cout << "Enter whole part: ";
    cin >> wp;
    cout << "Enter fraction part: ";
    cin >> fp;
    init(a, wp, fp);
}

void add(Fraction &a, Fraction &b)
{
    Fraction c;
    init(c, b.whole_part, b.fraction_part);
    fractional_cast(a, c);
    a.whole_part += c.whole_part;
    fraction_reduction(a);
}
```

```
void sub(Fraction &a, Fraction &b)
{
   Fraction c;
   init(c, b.whole_part, b.fraction_part);
   fractional_cast(a, c);
   a.whole_part -= c.whole_part;
   fraction_reduction(a);
}

void mul(Fraction &a, Fraction &b)
{
   a.whole_part *= b.whole_part;
   a.fraction_part *= b.fraction_part;
   fraction_reduction(a);
}

void equal(Fraction &a, Fraction &b)
{
   Fraction c, d;
   init(c, a.whole_part, a.fraction_part);
   init(d, b.whole_part, b.fraction_part);
   fractional_cast(c, d);
   if (c.whole_part > d.whole_part)
   {
       fraction_reduction(c); fraction_reduction(d);
       cout << c.whole_part << "/" << c.fraction_part << " is bigger
that " << d.whole_part << "/" << d.fraction_part << "\n";
   }
   else if (c.whole_part < d.whole_part)
   {
       fraction_reduction(c); fraction_reduction(d);
       cout << d.whole_part << "/" << d.fraction_part << " is bigger
that " << c.whole_part << "/" << c.fraction_part << "\n";
   }
   else
   {
       fraction_reduction(c); fraction_reduction(d);
       cout << c.whole_part << "/" << c.fraction_part << " Equal " <<
d.whole_part << "/" << d.fraction_part << "\n";
   }
}
```

```
int main()
{
   Fraction tests[2];
   for (int i = 0; i < 2; i++)
   {
       read(tests[i]);
   }
   add(tests[0], tests[1]);
   display(tests[0]);
   sub(tests[1], tests[0]);
   display(tests[1]);
   mul(tests[0], tests[1]);
   display(tests[0]);
   equal(tests[0], tests[1]);
   equal(tests[0], tests[0]);
   return 0;
}
```

Результат.

```
Enter whole part: 1
Enter fraction part: 2
Enter whole part: 1
Enter fraction part: 4
3/4
-1/2
-3/8
-3/8 is bigger that -1/2
-3/8 Equal -3/8
```