МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКОЙ ФЕДЕРАЦИИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ
ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

**«БЕЛГОРОДСКИЙ ГОСУДАРСТВЕННЫЙ
ТЕХНОЛОГИЧЕСКИЙ УНИВЕРСИТЕТ им. В. Г. ШУХОВА»
(БГТУ им. В.Г. Шухова)**

Кафедра программного обеспечения вычислительной техники и
автоматизированных систем

Лабораторная работа № 7

по дисциплине: «Объектно-ориентированное
программирование»

Выполнил: ст. группы ПВ-211
Медведев Дмитрий Сергеевич

Проверил:
Буханов Дмитрий Геннадьевич
Харитонов Сергей Дмитриевич

Белгород 2023 г.

Исключительные ситуации в C++.

Вариант 8

Цель работы: Получение теоретических знаний об исключительных ситуациях в C++. Получение практических навыков при работе с исключениями в C++.

Задание:

1. Изучить теоретические сведения об исключениях в C++.
2. Изучить самостоятельно стандартные классы для исключений предусмотренных в C++.
3. Разработать программу в соответствии с заданным вариантом задания.
4. Оформить отчет.

Разработать класс для работы с дробями. Переопределить операции. Сделать возможность работы с неправильными дробями. Предусмотреть исключительные ситуации (деление на ноль, переполнение).

```
#include <iostream>

struct DivisionByZeroException : std::exception {
    float divider;

    explicit DivisionByZeroException(float value) {
        divider = value;
    }
};

struct Overflow : std::exception {
    long long value;

    explicit  Overflow(long long Value) {
        value = Value;
    }
};

struct fraction {
    fraction() {
        Numenator = 0;
        Denuminator = 1;
    }

    fraction(int numerator, int denominator) {
        Numenator = numerator;
```

```
        Denuminator = denominator;
    }

    fraction(int numerator) {
        Numenator = numerator;
        Denuminator = 1;
    }

    int Numenator;
    int Denuminator;

    friend fraction operator+(const fraction &x, const fraction &y) {
        long long factor = lcm(x.Denuminator, y.Denuminator);
        long long newNumerator = x.Numenator * factor / x.Denuminator +
y.Numenator * factor / y.Denuminator;
        if (factor > INT32_MAX || factor < INT32_MIN) {
            throw Overflow(factor);
        }
        if (newNumerator > INT32_MAX || newNumerator < INT32_MIN) {
            throw Overflow(factor);
        }

            fraction F = *new fraction((int) newNumerator, (int) factor);

        return reduction(F);
    };

    friend fraction operator-(const fraction &x, const fraction &y) {
        if (y.Numenator != 0) {
            long long factor = lcm(x.Denuminator, y.Denuminator);
            long long newNumerator = (long long) x.Numenator * factor /
x.Denuminator - y.Numenator * factor / y
                    .Denuminator;

            if (factor > INT32_MAX || factor < INT32_MIN) {
                throw Overflow(factor);
            }
            if (newNumerator > INT32_MAX || newNumerator < INT32_MIN) {
                throw Overflow(factor);
            }

            fraction F = *new fraction(newNumerator, factor);

            return reduction(F);
        } else
            return x;
    };

    friend fraction operator/(const fraction &x, const fraction &y) {
        if (x.Denuminator * y.Numenator == 0) {
            throw DivisionByZeroException((float) (x.Denuminator *
y.Numenator));
        }
```

```
        long long newNumerator = (long long) x.Numenator * y.Denuminator;
        long long newDenuminator = (long long) x.Denuminator * y.Numenator;

        if (newDenuminator > INT32_MAX || newDenuminator < INT32_MIN) {
            throw Overflow(newDenuminator);
        }
        if (newNumerator > INT32_MAX || newNumerator < INT32_MIN) {
            throw Overflow(newNumerator);
        }

        fraction F = *new fraction((int) newNumerator, (int)
newDenuminator);

        return reduction(F);
    };

    friend fraction operator*(const fraction &x, const fraction &y) {

        long long newNumerator = (long long) x.Numenator * y.Numenator;
        long long newDenuminator = (long long) x.Denuminator *
y.Denuminator;

        if (newDenuminator > INT32_MAX || newDenuminator < INT32_MIN) {
            throw Overflow(newDenuminator);
        }
        if (newNumerator > INT32_MAX || newNumerator < INT32_MIN) {
            throw Overflow(newNumerator);
        }

        fraction F = *new fraction(newNumerator, newDenuminator);

        return reduction(F);
    };

    friend bool operator==(const fraction &x, const fraction &y) {
        return x.Numenator == y.Numenator && x.Denuminator ==
y.Denuminator;
    }

    friend bool operator!=(const fraction &x, const fraction &y) {
        return x.Numenator != y.Numenator || x.Denuminator !=
y.Denuminator;
    }

    friend bool operator>(const fraction &x, const fraction &y) {
        int factor = lcm(x.Denuminator, y.Denuminator);
        fraction F1 = *new fraction(x.Numenator * factor / x.Denuminator,
factor);
        fraction F2 = *new fraction(y.Numenator * factor / y.Denuminator,
factor);

        return F1.Numenator > F2.Numenator;
    }

    static fraction reduction(fraction F) {
```

```
        int factor = gcd(F.Numenator, F.Denuminator);

        F.Numenator /= factor;
        F.Denuminator /= factor;

        if (F.Denuminator < 0) {
            F.Denuminator *= -1;
            F.Numenator *= -1;
        }

        return F;
    }

    static long long gcd(long long a, long long b) {
        a = abs(a);
        b = abs(b);

        while (b != 0) {
            long long t = b;
            b = a % b;
            a = t;
        }
        return a;
    }

    static long long lcm(long long a, long long b) {
        a = abs(a);
        b = abs(b);

        return (long long) (a * b) / (long long) gcd(a, b);
    }

};
```

Пример на деление на ноль:

```
int main() {
    fraction f1(5, 3);
    fraction f2(0, 3);
    fraction f3 = f1 / f2;
}
```

```
libc++abi: terminating with uncaught exception of type DivisionByZeroException: std::exception

Process finished with exit code 134 (interrupted by signal 6: SIGABRT)
```

Пример на переполнение:

```
int main() {
    fraction f1(100000000, 3);
    fraction f2(100000000, 3);
    fraction f3 = f1 * f2;
}
```

```
libc++abi: terminating with uncaught exception of type Overflow: std::exception

Process finished with exit code 134 (interrupted by signal 6: SIGABRT)
```

Вывод: в ходе лабораторной работы мы получили теоретические знания об исключительных ситуациях в C++, а также практические навыки работы с исключениями в C++.