МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ МОСКОВСКИЙ АВИАЦИОННЫЙ ИНСТИТУТ
(НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСТИТЕТ)

ЛАБОРАТОРНАЯ РАБОТА №0.2 по курсу

объектно-ориентированное программирование I семестр, 2021/22 уч. год

Студент *Васютинский Вадим Александрович М80-208Б-20*
Преподаватель *Дорохов Евгений Павлович*

Цель:

- Изучение основ работы с классами в C++;
- Перегрузка операций и создание литералов

Требования к программе

Разработать программу на языке C++ согласно варианту задания. Программа на C++ должна собираться с помощью системы сборки CMake. Программа должна получать данные из стандартного ввода и выводить данные в стандартный вывод.

Реализовать над объектами реализовать в виде перегрузки операторов.

Реализовать пользовательский литерал для работы с константами объектов созданного класса.

6. Создать класс Rational для работы с рациональными числами.

Описание программы

Исходный код лежит в файле:

main.cpp - исполняемый код.

Дневник отладки

Во время выполнения лабораторной работы программа не нуждалась в отладке, все ошибки компиляции были исправлены с первой попытки.

После их исправления программа работала так, как было задумано изначально.

Недочёты

Недочётов не было обнаружено.

Выводы

В процессе выполнения работы я на практике познакомился с пользовательскими литералами. Это очень удобная и практическая вещь, о которой я не знал до курса ООП. Использование этого средства позволяет получать из заданных типов данных какие то данные, вычислять что то, без использования функций, а с помощью переопределения специального оператора

Исходный код

main.cpp

```
#include <iostream>
#include <exception>
#include <string>
#include <vector>
#include <sstream>

class Rational {
public:
    Rational(int a, int b): a(a), b(b) {
        if (b == 0) {
            b = 1;
        }
        reduce();}
    Rational(Rational& other): a(other.a), b(other.b) {reduce();}
    void reduce() {
        int c = a;
        int d = b;
        while (c != d) {
            if (c > d) {
                c -= d;
            } else {
                d -= c;
            }
        }
        a /= d;
        b /= d;
    }
    void add(Rational& r) {
        a = a * r.b + b * r.a;
        b *= r.b;
        reduce();
    }
    void sub(Rational& r) {
        a = a * r.b - b * r.a;
        b *= r.b;
        reduce();
    }
    void mul(Rational& r) {
        a *= r.a;
        b *= r.b;
        reduce();
    }
    void div(Rational& r) {
        a *= r.b;
```

```
        b *= r.a;
        reduce();
    }
    int a, b;
};

std::ostream& operator<<(std::ostream& os, const Rational& rat) {
    std::cout << rat.a << "/" << rat.b << std::endl;
    return os;
}

bool operator>(Rational& r, Rational& l) {
    return r.a * l.b > l.a * r.b;
}

bool operator<(Rational& r, Rational& l) {
    return r.a * l.b < l.a * r.b;
}

Rational operator+(Rational& r, Rational& l) {
    Rational newNum(r.a * l.b + l.a * r.b, r.b * l.b);
    return newNum;
}

Rational operator-(Rational& r, Rational& l) {
    Rational newNum(r.a * l.b - l.a * r.b, r.b * l.b);
    return newNum;
}

Rational operator*(Rational& r, Rational& l) {
    Rational newNum(r.a * l.a, r.b * l.b);
    return newNum;
}

Rational operator/(Rational& r, Rational& l) {
    Rational newNum(r.a * l.b, r.b * l.a);
    return newNum;
}

Rational operator "" _rat(const char* str, size_t size) {
    std::stringstream test(str);
    std::string segment;
    std::vector<std::string> seglist;

    while(std::getline(test, segment, '/'))
```

```
    {
        seglist.push_back(segment);
    }
    if (seglist.size() == 1) {
        Rational newRat(std::stoi(seglist[0]), 1);
        return newRat;
    }
    else {
        Rational newRat(std::stoi(seglist[0]), std::stoi(seglist[1]));
        return newRat;
    }
}

int main() {
    Rational a(1, 2);
    Rational b(1, 3);
    a.add(b);
    std::cout << a << std::endl;
    std::cout << (a+b) << std::endl;
    std::cout << "1/2"_rat << std::endl;
    return 0;
}
```