Московский Авиационный Институт
(Национальный Исследовательский Университет)

Кафедра 806 «Вычислительная информатика и программирование»
Факультет: «Информационные технологии и прикладная математика»

Лабораторная работа
Дисциплина: «Объектно-ориентированное программирование»
I I семестр
Задание 2: «Операторы, литералы»

Группа:	М8О-208Б-18, №3
Студент:	Анисимов Валерий Алексеевич
Преподаватель:	Журавлёв Андрей Андреевич
Оценка:	
Дата:	30.09.2019

Москва, 2019

1. **Тема**: Операторы, литералы

2. **Цель работы**: Извучение механизмов перегрузки операторов. Изучение механизмов работы с пользовательскими лиетралами.

3. **Задание** (*вариант № 3*):

Рациональная (несократимая) дробь представляется парой целых чисел (a, b), где a — числитель, b — знаменатель. Создать класс `Rational` для работы с рациональными дробями. Обязательно должны быть реализованы операции:

- сложения `add`, $(a, b) + (c, d) = (ad + bc, bd)$;
- вычитания `sub`, $(a, b) – (c, d) = (ad – bc, bd)$;
- умножения `mul`, $(a, b) \times (c, d) = (ac, bd)$;
- деления `div`, $(a, b) / (c, d) = (ad, bc)$;
- операции сравнения.

Должна быть реализована функция сокращения дроби reduce(), которая обязательно вызывается при выполнении арифметических операций.
Операции сложения, вычитания, умножения, деления, сравнения (на равенство, больше и меньше) должны быть выполнены в виде перегрузки операторов.
Необходимо реализовать пользовательский литерал для работы с константами типа `Rational`.

4. **Адрес репозитория на GitHub** https://github.com/wAlienUFOx/oop_exercise_02

5. **Код программы на C++**

main.cpp

```
#include <iostream>
#include "fraction.h"

int main() {
  fractions f;
  fractions f1;
  std::cout << "Введите первую дробь\n";
  std::cin >> f;
  std::cout << "Введите вторую дробь\n";
  std::cin >> f1;
  std::cout << "Первая дробь\n";
  std::cout << f << std::endl;
  std::cout << "Вторая дробь\n";
  std::cout << f1 << std::endl;
  std::cout << "Сумма\n";
  std::cout << f + f1 << std::endl;
  std::cout << "Разность\n";
  std::cout << f - f1 << std::endl;
  std::cout << "Произведение\n";
  std::cout << f * f1 << std::endl;
  std::cout << "Частное\n";
  std::cout << f / f1 << std::endl;
  if(f > f1)
    std::cout << "Первая дробь больше\n";
  if(f < f1)
    std::cout << "Первая дробь меньше\n";
  if(f == f1)
    std::cout << "Дроби равны\n";
  fractions f3;
  f3 = "[5:9]"_d;
  std::cout << f3 << std::endl;
```

```
  return 0;
}
```

fraction.h

```
#ifndef D_FRACTIONS_H
#define D_FRACTIONS_H

#include <iostream>

struct fractions{
  fractions();
  fractions(int a, int b);

  fractions& operator+= (const fractions& dr);
  fractions& operator-= (const fractions& dr);
  fractions& operator*= (const fractions& dr);
  fractions& operator/= (const fractions& dr);
  fractions operator+ (const fractions& dr) const;
  fractions operator- (const fractions& dr) const;
  fractions operator* (const fractions& dr) const;
  fractions operator/ (const fractions& dr) const;
  fractions _reduce ();
  bool operator> (const fractions& dr) const;
  bool operator< (const fractions& dr) const;
  bool operator== (const fractions& dr) const;
  friend std::istream& operator>> (std::istream& in, fractions& dr);
  friend std::ostream& operator<< (std::ostream& out, const fractions& dr);

public:
  int arr[2];

};

fractions operator ""_d(const char* str, size_t size);


#endif
```

fraction.cpp

```
#include "fraction.h"
#include <cstring>
#include <sstream>
#include <algorithm>

fractions::fractions(): arr{0, 0} {}
fractions::fractions(int a, int b): arr{a, b} {}

fractions& fractions::operator+= (const fractions& dr){
  fractions tmp;
  tmp.arr[0] = (arr[0] * dr.arr[1]) + (arr[1] * dr.arr[0]);
  tmp.arr[1] = arr[1] * dr.arr[1];
  tmp._reduce();
  arr[0] = tmp.arr[0];
  arr[1] = tmp.arr[1];
  return *this;
}
fractions& fractions::operator-= (const fractions& dr){
  fractions tmp;
```

```
  tmp.arr[0] = (arr[0] * dr.arr[1]) - (arr[1] * dr.arr[0]);
  tmp.arr[1] = arr[1] * dr.arr[1];
  tmp._reduce();
  arr[0] = tmp.arr[0];
  arr[1] = tmp.arr[1];
  return *this;
}
fractions& fractions::operator*= (const fractions& dr){
  fractions tmp;
  tmp.arr[0] = arr[0] * dr.arr[0];
  tmp.arr[1] = arr[1] * dr.arr[1];
  tmp._reduce();
  arr[0] = tmp.arr[0];
  arr[1] = tmp.arr[1];
  return *this;
}
fractions& fractions::operator/= (const fractions& dr){
  fractions tmp;
  tmp.arr[0] = arr[0] * dr.arr[1];
  tmp.arr[1] = arr[1] * dr.arr[0];
  tmp._reduce();
  arr[0] = tmp.arr[0];
  arr[1] = tmp.arr[1];
  return *this;
}

fractions fractions::operator+ (const fractions& dr) const{
  fractions result = *this;
  result += dr;
  return result;
}
fractions fractions::operator- (const fractions& dr) const{
  fractions result = *this;
  result -= dr;
  return result;
}
fractions fractions::operator* (const fractions& dr) const{
  fractions result = *this;
  result *= dr;
  return result;
}
fractions fractions::operator/ (const fractions& dr) const{
  fractions result = *this;
  result /= dr;
  return result;
}
fractions fractions::_reduce() {
  int g = std::__gcd(arr[0], arr[1]);
  arr[0] /= g;
  arr[1] /= g;
  return *this;
}
bool fractions::operator> (const fractions& dr) const{
  return ((arr[0] * dr.arr[1]) > (dr.arr[0] * arr[1]));
}
bool fractions::operator< (const fractions& dr) const{
  return ((arr[0] * dr.arr[1]) < (dr.arr[0] * arr[1]));
}
bool fractions::operator== (const fractions& dr) const{
  return ((arr[0] * dr.arr[1]) == (dr.arr[0] * arr[1]));
}
```

```
fractions operator ""_d(const char* str, size_t size){   //[5:9]
  std::istringstream is(str);
    char tmp;
    int c, z;
    is >> tmp >> c >> tmp >> z;
    return {c, z};
}


std::istream& operator>> (std::istream& in, fractions& dr){
  in >> dr.arr[0] >> dr.arr [1];
  return in;
}
std::ostream& operator<< (std::ostream& out, const fractions& dr){
  out << dr.arr[0] << '/' << dr.arr[1];
  return out;
}
```

CMakeLists.txt

```
cmake_minimum_required (VERSION 3.5)

project(lab2)

add_executable(oop_exercise_02
  main.cpp
  fraction.cpp)

set(CMAKE_CXX_FLAGS "${CMAKE_CXX_FLAGS} -Wall -Wextra")

set_target_properties(oop_exercise_02 PROPERTIES CXX_STANDART 14 CXX_STANDART_REQUIRED ON)
```

6. Набор testcases

test_01.txt	Ожидаемое действие	Ожидаемый результат
0 1 1 3	0/0 + 1/3	1/3
	0/0 - 1/3	-1/3
	0/0 * 1/3	0/0
	0/0 / 1/3	0/0
	0/0 < 1/3	1

	0/0 > 1/3	0
	0/0 == 1/3	0

test_02.txt	Ожидаемое действие	Ожидаемый результат
2 3 5 15	2/3 + 5/15	1/1
	2/3 - 5/15	1/3
	2/3 * 5/15	2/9
	2/3 / 5/15	2/1
	2/3 < 5/15	0
	2/3 > 5/15	1
	2/3 == 5/15	0

test_03.txt	Ожидаемое действие	Ожидаемый результат
16 10 16 10	16/10 + 16/10	16/5
	16/10 -16/10	0/0
	16/10 * 16/10	64/25
	16/10 / 16/10	1/1

16/10 < 16/10	0
16/10 > 16/10	0
16/10 == 16/10	1

7. Результаты выполнения тестов

```
walien@PC-name:~/2kurs/CPP/lab2/tmp$ ./oop_exercise_02 < ~/2kurs/CPP/lab2/test_01.txt
Введите первую дробь
Введите вторую дробь
Первая дробь
0/1
Вторая дробь
1/3
Сумма
1/3
Разность
-1/3
Произведение
0/1
Частное
0/1
Первая дробь меньше
5/9
walien@PC-name:~/2kurs/CPP/lab2/tmp$ ./oop_exercise_02 < ~/2kurs/CPP/lab2/test_02.txt
Введите первую дробь
Введите вторую дробь
Первая дробь
2/3
Вторая дробь
5/15
Сумма
1/1
Разность
1/3
Произведение
2/9
Частное
2/1
Первая дробь больше
5/9
walien@PC-name:~/2kurs/CPP/lab2/tmp$ ./oop_exercise_02 < ~/2kurs/CPP/lab2/test_03.txt
Введите первую дробь
Введите вторую дробь
Первая дробь
16/10
Вторая дробь
16/10
Сумма
16/5
Разность
```

0/1
Произведение
64/25
Частное
1/1
Дроби равны
5/9

8. Объяснение результатов работы программы - вывод

В fractions.h были заданы, а в fractions.cpp описаны, методы, операторы, литералы и свойства этого класса, применяемые в main.cpp.
Применение перегрузки операторов в классах может существенно облегчить и ускорить процесс написания кода, однако, при неосторожном обращении, может запутать код и затруднить его чтение.
Пользовательские литералы позволяют создавать объекты пользовательского типа посредством суффикса. Их использование может как повысить читаемость кода и упростить его написание, так и наоборот, при неумелом обращении.