Правительство Российской Федерации

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ
ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
«НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ
«ВЫСШАЯ ШКОЛА ЭКОНОМИКИ»

Отчёт к лабораторной работе №7

по дисциплине

«Языки программирования»

Работу выполнила

Студент группы СКБ222 _____ М. Х. Халимов

подпись, дата

Работу проверил _____ С. А. Булгаков

подпись, дата

Москва 2022

Содержание

Постановка задачи

Разработать класс *fraction* описывающий обыкновенную дробь. Интерфейс и реализацию разместить в файлах *fraction.h* и *fraction.cpp* соответственно. В качестве типов данных для хранения значений числителя и знаменателя использовать *unsigned long int*. Класс должен соответствовать позволять выполнять основные арифметические операции вида *f@f*, где *fraction f*, умножение/деление на целое число, приведение к (приближенному) значению типа *double*, а также операции помещения (извлечения) в поток (из потока). Формат ввода-вывода: *числитель/знаменатель*.

В основной функции, размещенной в файле *main.cpp*, продемонстрировать применение разработанного класса и его методов.

1. Описание класса *fraction*

Объявляется класс fraction, являющийся совокупностью полей numenator и denominator типа unsigned long int и методов fraction(), fraction(const fraction&), fraction(unsigned long, unsigned long), ~fraction(), operator=, operator+, operator-, operator*, operator/, operator double() const, operator<<, operator>>, simplify(). Класс отвечает за создание обыкновенной дроби.

2. Описание методов класса

2.1 Конструктор объекта класса *fraction*

Метод используется для создания объекта класса *fraction*. Метод создает *“пустой”* объект класса *fraction*.

2.2 Конструктор копирования

Метод используется для копирования объекта класса *fraction*.

2.3 Деструктор

Метод используется для уничтожения объекта класса *fraction.*

2.4 Конструктор объекта класса *fraction* при помощи двух переменных типа *unsigned long int*

Метод используется для создания объекта класса *fraction*. На вход метод получает две переменные типа *unsigned long int*. В результате создается объект класса *fraction*.

2.5 Метод перегрузки оператора “=”

Метод используется для перегрузки оператора *“=”*. На вход метод получает объект класса *fraction*. В результате происходит приравнивание объекта к некому *“значению”*.

2.6 Метод перегрузки оператора “+”

Метод используется для перегрузки оператора *“+”*. На вход метод получает два объекта класса *fraction*. В результате происходит сложение двух объектов класса fraction и возвращение нового объекта класса *fraction*.

2.7 Метод перегрузки оператора “-”

Метод используется для перегрузки оператора *“-”*. На вход получает два объекта класса *fraction*. В результате происходит нахождение разницы двух объектов класса *fraction* и возвращение нового объекта класса *fraction*.

2.8. Методы перегрузки оператора “*”

Метод используется для перегрузки оператора *“*”*. На вход получается два объекта класса *fraction* либо переменная *int* и объект класса *fraction*. В результате происходит перемножение двух чисел и возвращение нового объекта класса *fraction*.

2.9. Методы перегрузки оператора “/”

Метод используется для перегрузки оператора *“/”*. На вход получается два объекта класса *fraction* либо переменная *int* и объект класса *fraction*. В результате происходит деление двух чисел и возвращение нового объекта класса *fraction*.

2.10. Метод нахождения приближенного значения *double*

Метод используется для нахождения приближенного к получаемой на вход дроби значения.

2.11. Метод перегрузки оператора “<<”

Метод используется для перегрузки оператора *“<<”*. В результате объект класса *fraction* выводится из потока.

2.12. Метод перегрузки оператора “>>”

Метод используется для перегрузки оператора *“>>”*. В результате объект класса *fraction* вводится в поток.

2.13. Метод упрощения дроби

Метод используется для упрощения дробей, которые могут быть упрощены.

(Например, *3/6* будет упрощено как *1/2*)

3. Функция main

Функция *main* включает в себя создание объектов класса *fraction* и вызов методов этого же класса. В результате работы функции в консоль выводится сумма, разность, произведение, частное, получаемые из заданных чисел.

4. Результаты тестирования программы

Листинг

```
ISTREAM EXAMPLE
12
34
1
7
OSTREAM EXAMPLE
6/17 1/7
SUM EXAMPLE
a+b: 59/119
SUB EXAMPLE
a-b: 25/119
MUL EXAMPLE
a*b: 6/119
DIV EXAMPLE
a/b: 42/17
INT DIV EXAMPLE
1480/b: 10360/1
DIV WITH INT EXAMPLE
b/1480: 1/10360
INT MULLTIPLY EXAMPLE
1480*b: 1480/7
MULLTIPLY INT EXAMPLE
b*1480: 1480/7
DOUBLE EXAMPLE
0.142857
```

Приложение А

Исходный код

```
#include "fraction.h"

#include <iostream>
#include <algorithm>

fraction::fraction()
     : numenator(0), denominator(1) {}

fraction::fraction(const fraction& number)
     : numenator(number.numenator), denominator(number.denominator) {}

fraction::fraction(unsigned long a, unsigned long b)
     : numenator(a), denominator(b)
{
    if(denominator == 0)
    {
        std::cerr << "ERROR" << std::endl;
        denominator = 1;
    }
}

fraction::~fraction()
{
    numenator = 0;
    denominator = 1;
}

fraction fraction::operator=(const fraction& number)
{
    if (this == &number)
     {
        return *this;
    }

    this->numenator = number.numenator;
    this->denominator = number.denominator;

     return *this;
}

const fraction operator+(const fraction& left, const fraction& right)
{
    fraction left_clone = left, right_clone = right;

    unsigned                long                lcm                =
(left.denominator*right.denominator)/std::__gcd(left.denominator,
right.denominator);
```

```
    unsigned long left_coefficient = lcm/left.denominator,
right_coefficient = lcm/right.denominator;

     left_clone = fraction(left_clone.numenator*left_coefficient,
left_clone.denominator*left_coefficient);
    right_clone = fraction(right_clone.numenator*right_coefficient,
right_clone.denominator*right_coefficient);

     fraction task(left_clone.numenator + right_clone.numenator,
left_clone.denominator);

     task.simplify();

    return task;
}

const fraction operator-(const fraction& left, const fraction& right)
{

    fraction left_clone = left, right_clone = right;

     unsigned long lcm =
(left.denominator*right.denominator)/std::__gcd(left.denominator,
right.denominator);
    unsigned long left_coefficient = lcm/left.denominator,
right_coefficient = lcm/right.denominator;

     left_clone = fraction(left_clone.numenator*left_coefficient,
left_clone.denominator*left_coefficient);
    right_clone = fraction(right_clone.numenator*right_coefficient,
right_clone.denominator*right_coefficient);

     fraction task;

     if(left_clone.numenator >= right_clone.numenator)
     {
        task = fraction(left_clone.numenator - right_clone.numenator,
left_clone.denominator);
    }
     else
     {
        task = fraction(right_clone.numenator - left_clone.numenator,
left_clone.denominator);
    }

     task.simplify();

     return task;
}

const fraction operator*(const fraction& left, const fraction& right)
{
```

```
    fraction                        task(left.numenator*right.numenator,
left.denominator*right.denominator);

     task.simplify();

     return task;
}

const fraction operator*(int coefficient, const fraction& number)
{
    fraction task = number;

     task.numenator *= coefficient;
    task.simplify();

     return task;
}

const fraction operator*(const fraction& number, int coefficient)
{
    fraction task = number;

     task.numenator *= coefficient;
    task.simplify();

     return task;
}

const fraction operator/(const fraction& left, const fraction& right)
{
    fraction                      task(left.numenator*right.denominator,
left.denominator*right.numenator);

     task.simplify();

     return task;
}

const fraction operator/(int coefficient, const fraction& number)
{
    fraction task(coefficient*number.denominator, number.numenator);

     task.simplify();

     return task;
}

const fraction operator/(const fraction& number, int coefficient)
{
    if(coefficient != 0)
    {
```

```
        fraction                                    task(number.numenator,
number.denominator*coefficient);

          task.simplify();

          return task;
    }

     std::cerr << "ERROR" << std::endl;

     return fraction(0, 1);
}

std::ostream&  operator<<(std::ostream  &outstream,  const  fraction&
number)
{
    outstream << number.numenator << '/' << number.denominator;

     return outstream;
}

std::istream& operator>>(std::istream &inputstream, fraction& number)
{
    inputstream >> number.numenator;

     inputstream.ignore(1,'/');

     inputstream >> number.denominator;

     number.simplify();

     return inputstream;
}

void fraction::simplify()
{
    unsigned long simplified = std::__gcd(numenator, denominator);

     numenator /= simplified;
    denominator /= simplified;
}

fraction::operator double() const
{
    return double(numenator)/double(denominator);
}
```

Приложение Б