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 *BigFraction*, являющийся наследником класса *fraction* из лабораторной работы №7, использующий для хранения значений числителя и знаменателя класс *bigint* из лабораторной работы №6.
В основной функции, размещенной в файле *main.cpp*, продемонстрировать применение разработанного класса и его методов.

1. Описание класса Big*Fraction*

Идентичен классу fraction из Лабораторной работы №7, за исключением измененных полей. Вместо *unsigned long int* используется *bigint*.

2. Описание методов класса

2.1 Метод упрощения дроби

Метод используется для упрощения дробей, которые могут быть упрощены. Для упрощения используется метод *gcd* из Лабораторной работы №6 (Например, *3/6* будет упрощено как *1/2*)

3. Описание класса *bigint*

Идентичен своей версии из Лабораторной работы №6, за исключением добавленной перегрузки операторов +, -, /, =, *, ==, <<, >>.

4. Функция main

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

5. Результаты тестирования программы

Листинг

```
ISTREAM EXAMPLE
67897890768907657890756789075667890689657896577896578
78
75678098707890790790909007907564789080765467898795646789657890796578987
796757890756
3
OSTREAM EXAMPLE
2923923098230981923091292309122923026658599819599981/92
91292036209230996563636332309128599620981885992991288599819230998192369
232919230912/1
SUM EXAMPLE
a+b:
34604113564758091836376401483757119444760759972976585088056459170564536
477564591536/92
SUB EXAMPLE
```

a-b:
95045542788050353275532644300319548021698751690088185088056459170564536477564591536/92
MUL EXAMPLE
a*b:
84989622114071809139085574530809179088127947010490113776313291608840750408967014853713203636535918552913540381490145001961195137061 4/92
DIV EXAMPLE
a/b:
29239230982309819230912923091229230266585998195998 1/153748726714591725064545773915388782871847508869773850880564591705645364775645915 36
INT DIV EXAMPLE
1480/b:
3/91292036209230996563636332309128599620981885992991288599819230998192362329192309 12
DIV WITH INT EXAMPLE
b/1480:
91292036209230996563636332309128599620981885992991288599819230998192362329192309 12/3
INT MULLTIPLY EXAMPLE
1480*b:
75678098707890790790909007907564789080765467898795646789657890796578907567/1
MULLTIPLY INT EXAMPLE
b*1480:
75678098707890790790909007907564789080765467898795646789657890796578907567/1

Приложение А

Исходный код

```
#include "BigFraction.h"
#include "bigint.h"

#include <iostream>
#include <algorithm>

BigFraction::BigFraction()
     : numenator(bigint("0")), denominator(bigint("1")) {}

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

BigFraction::BigFraction(const bigint& numen, const bigint& denomin)
     : numenator(numen), denominator(denomin)
{
    if(denominator == bigint("0"))
    {
        std::cerr << "ERROR" << std::endl;
        exit(1);
    }
}

BigFraction::~BigFraction()
{
    numenator = bigint("0");
    denominator = bigint("1");
}

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

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

     return *this;
}

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

```
    bigint                                  lcm                                  =
bigint::div(bigint::mul(left.denominator,right.denominator),bigint::gc
d(left.denominator, right.denominator));
    bigint   left_coefficient   =   bigint::div(lcm,left.denominator),
right_coefficient = bigint::div(lcm,right.denominator);

    left_clone                                                               =
BigFraction(bigint::mul(left_clone.numenator,left_coefficient),
bigint::mul(left_clone.denominator,left_coefficient));
    right_clone                                                              =
BigFraction(bigint::mul(right_clone.numenator,right_coefficient),
bigint::mul(right_clone.denominator,right_coefficient));

    BigFraction                task(bigint::add(left_clone.numenator,
right_clone.numenator),  left_clone.denominator);

    task.simplify();

    return task;
}

const   BigFraction   operator-(const   BigFraction&   left,   const
BigFraction& right)
{
    BigFraction left_clone = left, right_clone = right;

    bigint                                  lcm                                  =
bigint::div(bigint::mul(left.denominator,right.denominator),bigint::gc
d(left.denominator, right.denominator));
    bigint   left_coefficient   =   bigint::div(lcm,left.denominator),
right_coefficient = bigint::div(lcm,right.denominator);

    left_clone                                                               =
BigFraction(bigint::mul(left_clone.numenator,left_coefficient),
bigint::mul(left_clone.denominator,left_coefficient));
    right_clone                                                              =
BigFraction(bigint::mul(right_clone.numenator,right_coefficient),
bigint::mul(right_clone.denominator,right_coefficient));

    BigFraction task;

    if(left_clone.numenator >= right_clone.numenator)
    {
       task      =      BigFraction(bigint::sub(left_clone.numenator,
right_clone.numenator),  left_clone.denominator);
    }
    else
    {
       task      =      BigFraction(bigint::sub(right_clone.numenator,
left_clone.numenator),  left_clone.denominator);
    }
```

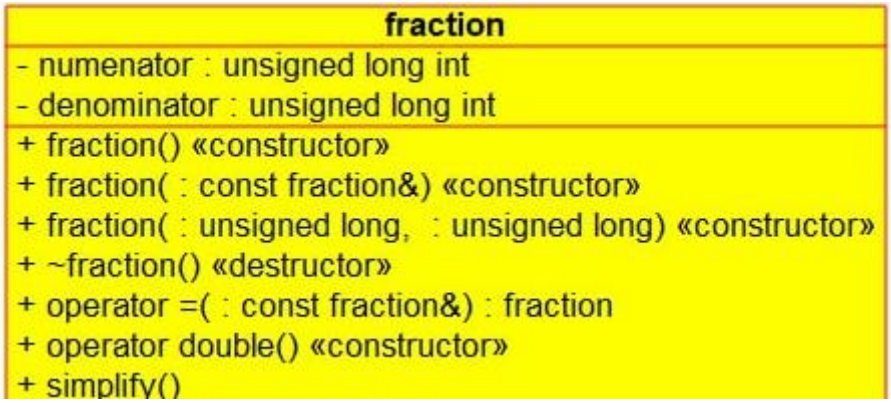
```
    task.simplify();

    return task;
}

const BigFraction operator*(const BigFraction& left, const
BigFraction& right)
{
    BigFraction task(bigint::mul(left.numenator,right.numenator),
bigint::mul(left.denominator,right.denominator));

    task.simplify();

    return task;
}

const BigFraction operator*(unsigned long coefficient, const
BigFraction& number)
{
    BigFraction task = number;

    task.numenator = bigint::mul(task.numenator,
bigint(coefficient));

    task.simplify();

    return task;
}

const BigFraction operator*(const BigFraction& number, unsigned long
coefficient)
{
    BigFraction task = number;

    task.numenator = bigint::mul(task.numenator,
bigint(coefficient));

    task.simplify();

    return task;
}

const BigFraction operator/(const BigFraction& left, const
BigFraction& right)
{
    BigFraction task(bigint::mul(left.numenator,right.denominator),
bigint::mul(left.denominator,right.numenator));

    task.simplify();

    return task;
}
```

```
const BigFraction operator/(unsigned long coefficient, const BigFraction& number)
{
    BigFraction task(bigint::mul(bigint(coefficient),number.denominator), number.numenator);

     task.simplify();

     return task;
}

const BigFraction operator/(const BigFraction& number, unsigned long coefficient)
{
    if (coefficient != 0)
    {
        bigint d = number.denominator;

          BigFraction task(number.numenator, bigint::mul(d, bigint(coefficient)));

          task.simplify();

          return task;
    }

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

     return BigFraction(bigint("0"), bigint("1"));
}

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

    return outstream;
}

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

     number.simplify();

     return inputstream;
}

void BigFraction::simplify()
```

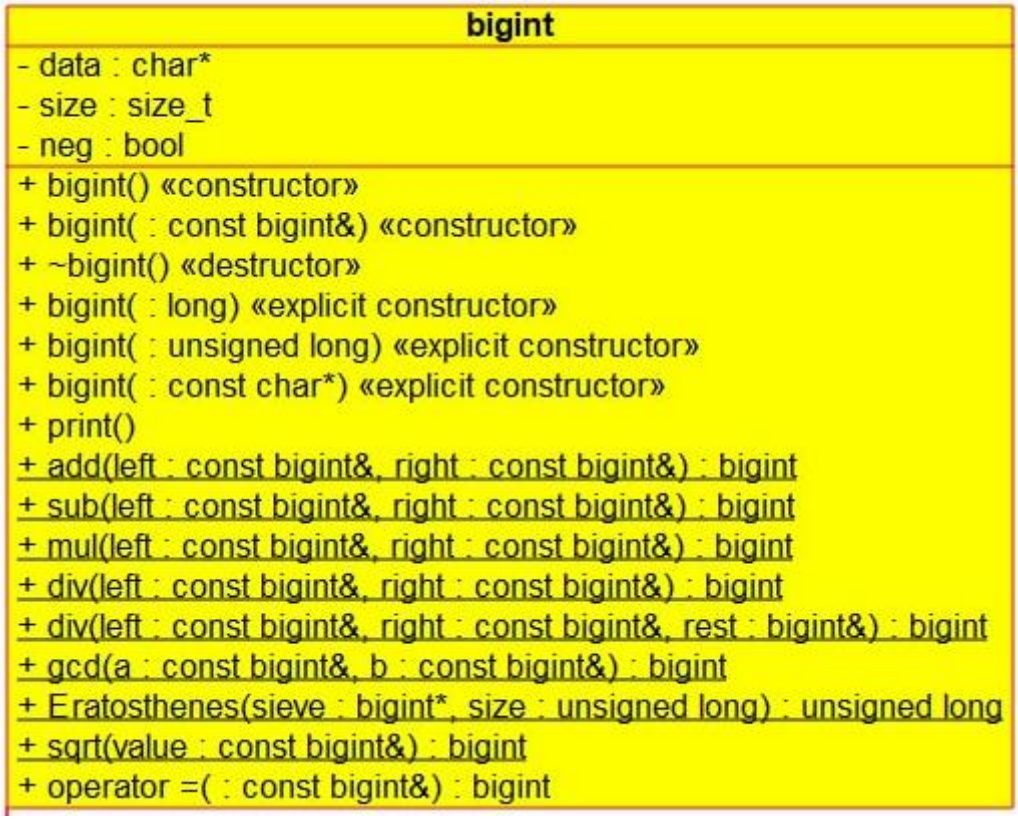
```
{
    bigint g = bigint::gcd(numenator, denominator);

     numenator = bigint::div(numenator, g);

     denominator = bigint::div(denominator, g);
}

/*BigFraction::operator double() const
{
     bigint reminder(numenator);

     double task;

     for(size_t index = 0; index <= 5; ++index)
     {
          task    +=    static_cast<int>(div(reminder,denominator))    /
pow(10.f, index);

          reminder = (reminder % denominator) * bigint(10);
     }

     return task;
}*/

//штука выше не работает, не придмула в итоге как заставить работать

//Я в итоге не понял как нужно написать double();
```

Приложение Б

UML-схема класса *fraction*



UML-схема класса *bigint*



UML-схема класса *BigFraction*