**Министерство науки и высшего образования Российской Федерации**
**Федеральное государственное бюджетное образовательное учреждение высшего образования**
**«Московский государственный технический университет имени Н. Э. Баумана**
**(национальный исследовательский университет)»**
**(МГТУ им. Н. Э. Баумана)**

ФАКУЛЬТЕТ «Информатика и системы управления» (ИУ)
КАФЕДРА «Информационная безопасность» (ИУ8)

# Домашнее задание № 2

# ПО КУРСУ

# «Алгоритмические языки»

# на тему «Объектно-ориентированные возможности языка Си++»

Студент ИУ8-25 (Группа) В.В.Гоза (И. О. Фамилия)

Преподаватель: В. В. Соборова (И.О. Фамилия)

2022 г.

## Условие:

Вариант 3

Определить класс CRats для работы с вектором несократимых дробей вида Pi/qi, где *Pi* — целое, *qi* -— натуральное.

Внутри класса вектор должен быть реализован с помощью указателей (одного или двух). Длина вектора задается в конструкторе класса и изменятся, если происходит присваивание вектору другой длины.

В классе должны быть определены необходимые конструкторы (в том числе конструктор копирования и перемещения), деструктор, операторы присваивания (копированием и перемещением), сложения, вычитания, << (вставка в поток вывода), инкремент и декремент ++ и -- (справа и слева), увеличивающие и уменьшающие длину вектора.

При сложении и вычитании длина результата - это минимум из длин исходных векторов.

Каждый класс должен быть описан в 2-х файлах (заголовочном и реализации). В отдельном файле должен быть написан тест на данный класс (функция main).

## Программа:

## CRats.h:

```
#ifndef DZ22_CRATS_H
#define DZ22_CRATS_H

#include <iostream>

class CRats {
    //double *Numerator = nullptr;
    //double *Denominator = nullptr;
    double *Result = nullptr;
    size_t Len = 0;
public:

    /// конструктор по умолчанию
    CRats();

    /// конструктор копирования
    CRats(const CRats &vec);

    /// конструктор перемещения
    CRats(CRats &&vec);

    /// пользовательский конструктор
    explicit CRats(const double *c, size_t len);

    CRats(double *Result, size_t Len);

    /// деструктор
    ~CRats();

    /// оператор присваивания копированием
    CRats &operator=(const CRats &vec);

    /// оператор присваивания перемещением
```

```
    CRats &operator=(CRats &&vec);

    /// префикный инкремент ++ увеличивающий длину вектора
    CRats &operator++();

    /// постфиксный инкремент ++ увеличивающий длину вектора
    CRats operator++(int d);

    /// префикный декремент -- уменьшающий длину вектора
    CRats &operator--();

    /// префикный декремент -- уменьшающий длину вектора
    CRats operator--(int d);

    /// оператор сложения
    CRats &operator+(CRats &vec);

    /// оператор вычитания
    CRats &operator-(CRats &vec);

    /// вставка в поток вывода
    friend std::ostream &operator<<(std::ostream &out, const CRats &vec);
};

/*/// оператор сложения
CRats operator+(const CRats &e);

///оператор вычитания
CRats operator-(const CRats &e);*/

/// вставка в поток вывода
std::ostream &operator<<(std::ostream &out, const CRats &vec);

#endif //DZ22_CRATS_H
```

## CRats.cpp:

```
#include "CRats.h"

CRats::CRats(double *Result, size_t Len) {
    this->Len = Len;
    this->Result = new double[Len];
    for (size_t i = 0; i < Len; ++i) {
        this->Result[i] = Result[i];
    }
}

/// конструктор по умолчанию
CRats::CRats() :Result(nullptr), Len(0) {}

/// конструктор копирования
CRats::CRats(const CRats &vec) : Len(vec.Len) {
    Result = new double [vec.Len];
    for (size_t i = 0; i < vec.Len; ++i) {
        Result[i] = vec.Result[i];
    }
}

/// конструктор перемещения
CRats::CRats(CRats &&vec) {
    std::swap(Len, vec.Len);
```

```
    std::swap(Result, vec.Result);
}

///пользовательский конструктор
CRats::CRats(const double *c, const size_t len) : Len(len) {
    new int[len];
    for (size_t i = 0; i < len; ++i) {
        Result[i] = c[i];
    }
}

///деструктор
CRats::~CRats() {
    Len = 0;
    //delete[] Numerator;
    //delete[] Denominator;
    delete[] Result;
}

/// оператор присваивания копированием
CRats &CRats::operator=(const CRats &vec) {
    if (this != &vec) {
        Len = vec.Len;
        Result = new double [vec.Len];
        for (size_t i = 0; i < vec.Len; ++i) {
            Result[i] = vec.Result[i];
        }
    }
    return *this;
}

/// конструктор перемещения
CRats &CRats::operator=(CRats &&vec) {
    std::swap(Len, vec.Len);
    std::swap(Result, vec.Result);
    return *this;
}

/// вставка в поток вывода
std::ostream &operator<<(std::ostream &out, const CRats &vec) {
    out << vec.Result[0];
    for (size_t i = 1; i < vec.Len; ++i) {
        out << ' ' << vec.Result[i];
    }
    out << std::endl << "len: " << vec.Len;
    return out;
}

/// префикный инкремент ++ увеличивающий длину вектора
CRats &CRats::operator++() {
    ++Len;
    auto temp = new double[Len];
    for (size_t i = 0; i < Len - 1; ++i) {
        temp[i] = Result[i];
    }
    temp[Len] = 0;
    delete[] Result;
    Result = new double [Len];
    for (size_t i = 0; i < Len; ++i) {
        Result[i] = temp[i];
    }
    return *this;
}
```

```
/// постфиксный инкремент ++ увеличивающий длину вектора
CRats CRats::operator++(int d) {
    Len++;
    auto temp = new double [Len];
    for (size_t i = 0; i < Len - 1; ++i) {
        temp[i] = Result[i];
    }
    temp[Len] = 0;
    delete[] Result;
    Result = new double [Len];
    for (size_t i = 0; i < Len; ++i) {
        Result[i] = temp[i];
    }
    return *this;
}

/// префикный декремент -- уменьшающий длину вектора
CRats &CRats::operator--() {
    --Len;
    auto temp = new double [Len];
    for (size_t i = 0; i < Len; ++i) {
        temp[i] = Result[i];
    }
    delete[] Result;
    Result = new double [Len];
    for (size_t i = 0; i < Len; ++i) {
        Result[i] = temp[i];
    }
    return *this;
}

///постфиксный декремент -- уменьшающий длину вектора
CRats CRats::operator--(int d) {
    Len--;
    auto temp = new double [Len];
    for (size_t i = 0; i < Len; ++i) {
        temp[i] = Result[i];
    }
    delete[] Result;
    Result = new double [Len];
    for (size_t i = 0; i < Len; ++i) {
        Result[i] = temp[i];
    }
    return *this;
}

/// оператор сложения
CRats &CRats::operator+(CRats &vec) {
    if (Len > vec.Len) {
        auto *NewResult = new double [vec.Len];
        for (size_t i = 0; i < vec.Len; ++i) {
            NewResult[i] = Result[i];
        }
        delete[] Result;
        Len = vec.Len;
        Result = new double [vec.Len];
        for (size_t i = 0; i < vec.Len; ++i) {
            Result[i] = NewResult[i];
        }
        delete[] NewResult;
    } else if (Len < vec.Len) {
        auto *NewResult = new double [Len];
        for (size_t i = 0; i < Len; ++i) {
            NewResult[i] = vec.Result[i];
```

```
        }
        delete[] vec.Result;
        vec.Len = Len;
        vec.Result = new double [Len];
        for (size_t i = 0; i < Len; ++i) {
            vec.Result[i] = NewResult[i];
        }
        delete[] NewResult;
    }
    if (vec.Result != nullptr) {
        auto *temp = new double [Len];
        for (size_t i = 0; i < Len; ++i) {
            temp[i] = Result[i] + vec.Result[i];
        }
        delete[] Result;
        Result = new double [Len];
        for (size_t i = 0; i < Len; ++i){
            Result[i] = temp[i];
        }
        delete[] temp;
    }
    return *this;
}

///оператор вычитания
CRats &CRats::operator-(CRats &vec) {
    if (Len > vec.Len) {
        auto *NewResult = new double [vec.Len];
        for (size_t i = 0; i < vec.Len; ++i) {
            NewResult[i] = Result[i];
        }
        Len = vec.Len;
        delete[] Result;
        Result = new double [vec.Len];
        for (size_t i = 0; i < vec.Len; ++i) {
            Result[i] = NewResult[i];
        }
        delete[] NewResult;
    } else if (Len < vec.Len) {
        auto *NewResult = new double [Len];
        for (size_t i = 0; i < Len; ++i) {
            NewResult[i] = vec.Result[i];
        }
        vec.Len = Len;
        delete[] vec.Result;
        vec.Result = new double [Len];
        for (size_t i = 0; i < Len; ++i) {
            vec.Result[i] = NewResult[i];
        }
        delete[] NewResult;
    }
    if (vec.Result != nullptr) {
        auto *temp = new double [Len];
        for (size_t i = 0; i < Len; ++i) {
            temp[i] = Result[i] - vec.Result[i];
        }
        delete[] Result;
        Result = new double [Len];
        for (size_t i = 0; i < Len; ++i){
            Result[i] = temp[i];
        }
        delete[] temp;
    }
```

```
    return *this;
}
```

**main.cpp:**

```
#include "CRats.h"

int main() {

    auto *num = new double[7] {1, 2, 3, 4, 5, 6, 7};
    auto *denom = new double[7] {3, 3, 7, 7, 7, 7, 8};
    auto *num2 = new double[2] {3, 7};
    auto *denom2 = new double[2] {4, 9};
    std::cout << "\nVector №1" << std::endl;
    for (size_t i = 0; i < 7; ++i) {
        std::cout << num[i] << " ";
    }
    std::cout << std::endl;
    for (size_t i = 0; i < 7; ++i) {
        std::cout << denom[i] << " ";
    }
    std::cout << "\n\nVector of fraction №1" << std::endl;
    auto *C = new double[7];
    for (size_t i = 0; i < 7; ++i) {
        C[i] = num[i] / denom[i];
        std::cout << C[i] << " ";
    }
    std::cout << "\n\nOr alternative type of writing" << std::endl;
    for (size_t i = 0; i < 7; ++i) {
        std::cout << num[i] << "/" << denom[i] << " ";
    }
    std::cout << "\n\nVector №2" << std::endl;
    for (size_t i = 0; i < 2; ++i) {
        std::cout << num2[i] << " ";
    }
    std::cout << std::endl;
    for (size_t i = 0; i < 2; ++i) {
        std::cout << denom2[i] << " ";
    }
    std::cout << "\n\nVector of fraction №2" << std::endl;
    auto *C2 = new double[2];
    for (size_t i = 0; i < 2; ++i) {
        C2[i] = num2[i] / denom2[i];
        std::cout << C2[i] << " ";
    }
    std::cout << "\n\nOr alternative type of writing" << std::endl;
    for (size_t i = 0; i < 2; ++i) {
        std::cout << num2[i] << "/" << denom2[i] << " ";
    }
    std::cout << "\n\n=============\nConstructors: \n=============\n" <<
std::endl;
    CRats s1(C, 7);
    CRats s2(C2, 2);
    CRats s3;
    CRats s4;
    CRats s5(s1);
    CRats s6(s1);
    CRats s7(s1);
    CRats s8(s2);
    std::cout << "s1: " << s1 << std::endl << "s2: " << s2 << std::endl;
    std::cout << "\n================\nOperator =
```

```
(copy)\n=================\n" << std::endl;
    s3 = s1;
    std::cout << "s3: " << s3 << std::endl;
    std::cout << "\n=================\nOperator = (move)
\n=================\n" << std::endl;
    s4 = std::move(s1);
    std::cout << "s4: " << s4 << std::endl;
    std::cout << "\n===================================\nOperator ++ (after
right increment)\n===================================\n" << std::endl;
    s1++;
    std::cout << "s1: " << s1 << std::endl;
    std::cout << "\n===================================\nOperator ++ (after
left increment)\n===================================\n" << std::endl;
    ++s3;
    std::cout << "s3: " << s3 << std::endl;
    std::cout << "\n===================================\nOperator -- (after
right decrement) \n===================================\n" << std::endl;
    s4--;
    std::cout << "s4: " << s4 << std::endl;
    std::cout << "\n===================================\nOperator -- (after
left decrement) \n===================================\n" << std::endl;
    --s5;
    std::cout << "s5: " << s5 << std::endl;
    std::cout << "\n===========\nOperator + \n===========\n" << std::endl;
    std::cout << "-------------\ns3 = s6 + s2\n-------------\n" << std::endl;
    s3 = s6 + s2;
    std::cout << s3 << std::endl;
    std::cout << "\n===========\nOperator - \n===========\n" << std::endl;
    std::cout << "-------------\ns4 = s7 - s8\n-------------\n" << std::endl;
    s4 = s7 - s8;
    std::cout << s4 << std::endl;
    return 0;
}
```

**Вывод программы:**

```
Vector №1
1 2 3 4 5 6 7
3 3 7 7 7 7 8

Vector of fraction №1
0.333333 0.666667 0.428571 0.571429 0.714286 0.857143 0.875

Or alternative type of writing
1/3 2/3 3/7 4/7 5/7 6/7 7/8

Vector №2
3 7
4 9

Vector of fraction №2
0.75 0.777778

Or alternative type of writing
3/4 7/9

=============
Constructors:
=============

s1: 0.333333 0.666667 0.428571 0.571429 0.714286 0.857143 0.875
len: 7
s2: 0.75 0.777778
len: 2

=================
Operator = (copy)
=================

s3: 0.333333 0.666667 0.428571 0.571429 0.714286 0.857143 0.875
len: 7

=================
Operator = (move)
=================

s4: 0.333333 0.666667 0.428571 0.571429 0.714286 0.857143 0.875
len: 7

===================================
Operator ++ (after right increment)
===================================

s1: 0
len: 1
```

```
==================================
Operator ++ (after left increment)
==================================

s3: 0.333333 0.666667 0.428571 0.571429 0.714286 0.857143 0.875 0
len: 8

==================================
Operator -- (after right decrement)
==================================

s4: 0.333333 0.666667 0.428571 0.571429 0.714286 0.857143
len: 6

==================================
Operator -- (after left decrement)
==================================

s5: 0.333333 0.666667 0.428571 0.571429 0.714286 0.857143
len: 6

===========
Operator +
===========

-------------
s3 = s6 + s2
-------------

1.08333 1.44444
len: 2

===========
Operator -
===========

-------------
s4 = s7 - s8
-------------

-0.416667 -0.111111
len: 2
```