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»

КАФЕДРА «Теоретическая информатика и компьютерные технологии»

Лабораторная работа №7

по курсу «Языки и методы программирования»

«Разработка простейшего класса на C++»

Студент группы ИУ9-22Б Лавров Р. Д.

Преподаватель Посевин Д. П.

Москва 2025

1 Задание

Выриант 43: Матрица рациональных чисел размером m × n с операциями:

1. получение количества строк;

2. получение количества столбцов;

3. получение ссылки на указанный элемент;

4. умножение строки на рациональное число;

5. прибавление одной строки к другой.

Для представления рациональных чисел требуется реализовать класс нормализованных дробей с операциями сложения и умножения.

Выриант 49: Замкнутая ломаная линия на плоскости с операциями:

1. получение количества отрезков, из которых состоит ломаная;

2. получение ссылки на i-тую точку;

3. удаление отрезков, длина которых меньше указанного значения.

Точки на плоскости должны быть представлены структурами с вещественными полями x и y.

2 Результаты

Исходный код 9– 5.

Вариант 43

Листинг 1 — declaration.h (43 вариант)

```
#ifndef DECLARATION_H
#define DECLARATION_H

class Rational {
private:
    int numerator;
    int denominator;
    void normalize();
public:
    Rational(int num, int denom);
    Rational(const Rational& other);
    int getNumerator();
    int getDenominator();
    Rational operator+(Rational other);
    Rational operator*(Rational other);
    Rational& operator=(const Rational& other);
};


class RationalMatrix {
private:
    Rational** data;
    int rows;
    int cols;
public:
    RationalMatrix(int m, int n);
    RationalMatrix(const RationalMatrix& other);
    virtual ~RationalMatrix();
    int getRowCount();
    int getColCount();
    Rational& getElement(int i, int j);
    void multiplyRow(int row, const Rational& scalar);
    void addRow(int targetRow, int sourceRow, const Rational& scalar);
    RationalMatrix& operator=(const RationalMatrix& other);
};

#endif
```

Листинг 2 — implementation.cpp (43 вариант)

```
#include "declaration.h"
#include <algorithm>

int GCD(int a, int b) {
    while (b != 0) {
        int temp = b;
        b = a % b;
        a = temp;
    }
    return a;
}
```

Листинг 3 — implementation.cpp (43 вариант) (продолжение)

```
void Rational::normalize() {
    if (denominator < 0) {
        numerator *= -1;
        denominator *= -1;
    }
    int gcd = GCD(abs(numerator), abs(denominator));
    if (gcd != 0) {
        numerator /= gcd;
        denominator /= gcd;
    }
}

Rational::Rational(int num, int denom) : numerator(num), denominator(denom) {
    if (denominator == 0) {
        denominator = 1;
    }
    normalize();
}


Rational::Rational(const Rational& other)
    : numerator(other.numerator), denominator(other.denominator) {}

int Rational::getNumerator() { return numerator; }
int Rational::getDenominator() { return denominator; }

Rational Rational::operator+(Rational other) {
    int newNum = numerator * other.denominator + other.numerator * denominator;
    int newDenom = denominator * other.denominator;
    return Rational(newNum, newDenom);
}

Rational Rational::operator*(Rational other) {
    int newNum = numerator * other.numerator;
    int newDenom = denominator * other.denominator;
    return Rational(newNum, newDenom);
}

Rational& Rational::operator=(const Rational& other) {
    if (this != &other) {
        numerator = other.numerator;
        denominator = other.denominator;
    }
    return *this;
}

RationalMatrix::RationalMatrix(int m, int n) : rows(m), cols(n) {
    data = new Rational*[rows];
    for (int i = 0; i < rows; ++i) {
        data[i] = new Rational[cols];
    }
}
```

Листинг 4 — implementation.cpp (43 вариант) (продолжение 2)

```
RationalMatrix::RationalMatrix(const RationalMatrix& other) : rows(other.rows), cols(other.cols)
    {
    data = new Rational*[rows];
    for (int i = 0; i < rows; ++i) {
        data[i] = new Rational[cols];
        for (int j = 0; j < cols; ++j) {
            data[i][j] = other.data[i][j];
        }
    }
}

RationalMatrix::~RationalMatrix() {
    for (int i = 0; i < rows; ++i) {
        delete[] data[i];
    }
    delete[] data;
}

int RationalMatrix::getRowCount() { return rows; }
int RationalMatrix::getColCount() { return cols; }

Rational& RationalMatrix::getElement(int i, int j) {
    return data[i][j];
}

void RationalMatrix::multiplyRow(int row, const Rational& scalar) {
    for (int j = 0; j < cols; ++j) {
        data[row][j] = data[row][j] * scalar;
    }
}

void RationalMatrix::addRow(int targetRow, int sourceRow, const Rational& scalar) {
    for (int j = 0; j < cols; ++j) {
        data[targetRow][j] = data[targetRow][j] + (data[sourceRow][j] * scalar);
    }
}

RationalMatrix& RationalMatrix::operator=(const RationalMatrix& other) {
    if (this != &other) {
        for (int i = 0; i < rows; ++i) {
            delete[] data[i];
        }
        delete[] data;

        rows = other.rows;
        cols = other.cols;
        data = new Rational*[rows];
        for (int i = 0; i < rows; ++i) {
            data[i] = new Rational[cols];
            for (int j = 0; j < cols; ++j) {
                data[i][j] = other.data[i][j];
            }
        }
    }
    return *this;
}
```

Листинг 5 — main.cpp (43 вариант)

```cpp
#include <iostream>
#include "Polygon.cpp"

int main() {
    std::pair<double, double> square[4] = {
        {0.0, 0.0},
        {1.0, 0.0},
        {1.0, 1.0},
        {0.0, 1.0}
    };
    Polygon<4, double> poly(square);
    std::cout << "Perimeter: " << poly.perimeter() << std::endl;
    double pi = 2 * acos(0.0);
    poly.rotate(pi / 2);

    for (int i = 0; i < 4; ++i) {
        const auto& vertex = poly.getVertex(i);
        std::cout << "Vertex " << i << ": (" << vertex.first << ", " << vertex.second << ")"
 << std::endl;
    }
    return 0;
}
```

Вариант 49

Листинг 6 — declaration.h (49 вариант)

```cpp
#ifndef DECLARATION_H
#define DECLARATION_H

struct Point {
    double x;
    double y;
};

class Polyline {
private:
    Point* points;
    int segmentCount;
    int capacity;
public:
    Polyline();
    Polyline(const Polyline& other);
    virtual ~Polyline();
    Polyline& operator=(const Polyline& other);
    int getSegmentCount() const;
    Point& getPoint(int index);
    const Point& getPoint(int index) const;
    void addPoint(double x, double y);
    void removeShortSegments(double minLength);
};

#endif
```

Листинг 7 — implementation.cpp (49 вариант)

```cpp
#include "declaration.h"
#include <cmath>
#include <algorithm>

Polyline::Polyline() : points(nullptr), segmentCount(0), capacity(0) {}

Polyline::Polyline(const Polyline& other) : segmentCount(other.segmentCount), capacity(other.
    capacity) {
    points = new Point[capacity];
    for (int i = 0; i < segmentCount; ++i) {
        points[i] = other.points[i];
    }
}

Polyline::~Polyline() {
    delete[] points;
}

Polyline& Polyline::operator=(const Polyline& other) {
    if (this != &other) {
        delete[] points;
        segmentCount = other.segmentCount;
        capacity = other.capacity;
        points = new Point[capacity];
        for (int i = 0; i < segmentCount; ++i) {
            points[i] = other.points[i];
        }
    }
    return *this;
}

int Polyline::getSegmentCount() const {
    return segmentCount;
}

Point& Polyline::getPoint(int index) {
    return points[index];
}

const Point& Polyline::getPoint(int index) const {
    return points[index];
}

void Polyline::addPoint(double x, double y) {
    if (segmentCount >= capacity) {
        int newCapacity = (capacity == 0) ? 2 : capacity * 2;
        Point* newPoints = new Point[newCapacity];
        for (int i = 0; i < segmentCount; ++i) {
            newPoints[i] = points[i];
        }
        delete[] points;
        points = newPoints;
        capacity = newCapacity;
    }
    points[segmentCount] = {x, y};
    segmentCount++;
}
```

Листинг 8 — implementation.cpp (49 вариант) (продолжение)

```
void Polyline::removeShortSegments(double minLength) {
    if (segmentCount < 2) return;

    int newCount = 0;
    for (int i = 0; i < segmentCount; ++i) {
        int next = (i + 1) % segmentCount;
        double dx = points[next].x - points[i].x;
        double dy = points[next].y - points[i].y;
        double length = sqrt(dx*dx + dy*dy);

        if (length >= minLength) {
            points[newCount++] = points[i];
        }
    }

    segmentCount = newCount;
}
```

Листинг 9 — main.cpp (49 вариант)

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

void printPolyline(Polyline& pl) {
    std::cout << "Ломаная линия содержит " << pl.getSegmentCount() << " точек:" << std
    ::endl;
    for (int i = 0; i < pl.getSegmentCount(); ++i) {
        Point& p = pl.getPoint(i);
        std::cout << "Точка " << i << ": (" << p.x << ", " << p.y << ")" << std::endl;
    }
    std::cout << std::endl;
}

int main() {
    Polyline pl;
    pl.addPoint(0.0, 0.0);
    pl.addPoint(1.0, 1.0);
    pl.addPoint(5.0, 0.0);
    pl.addPoint(1.0, -1.0);

    std::cout << "Было:" << std::endl;
    printPolyline(pl);

    double minLength = 1.5;
    pl.removeShortSegments(minLength);
    std::cout << "После удаления" << minLength << ":" << std::endl;
    printPolyline(pl);

    std::cout << "точка изменена:" << std::endl;
    pl.getPoint(0).x = 10.0;
    pl.getPoint(0).y = 10.0;
    printPolyline(pl);

    return 0;
}
```

```
veter.ok77@MacBook-Pro-Rodion var1 % ./program
Исходная матрица:
1/2     3/4     5/6
7/8     9/10    11/12

После умножения строки 0 на 2:
1/1     3/2     5/3
7/8     9/10    11/12

После добавления половины строки 0 к строке 1:
1/1     3/2     5/3
11/8    33/20   7/4

Внутри функции testCopy (передача по значению):
1/1     3/2     5/3
11/8    33/20   7/4

Скопированная матрица (через оператор присваивания):
1/1     3/2     5/3
11/8    33/20   7/4
```

Рис. 1 — Результат работы

```
veter.ok77@MacBook-Pro-Rodion var2 % ./program
Исходная ломаная линия:
Ломаная линия содержит 4 точек:
Точка 0: (0, 0)
Точка 1: (1, 1)
Точка 2: (5, 0)
Точка 3: (1, -1)

После удаления отрезков короче 1.5:
Ломаная линия содержит 3 точек:
Точка 0: (1, 1)
Точка 1: (5, 0)
Точка 2: (1, -1)

Первая точка изменена на (10.0, 10.0):
Ломаная линия содержит 3 точек:
Точка 0: (10, 10)
Точка 1: (5, 0)
Точка 2: (1, -1)
```

Рис. 2 — Результат работы

3 Вывод

Я начал своё обучение языку C++