

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ
УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
«НОВОСИБИРСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»
Кафедра вычислительной техники

**Отчет по лабораторным работам №1-6
по дисциплине: «Программирование»**

Факультет: АВТФ
Группа: АВТ-143
Студент: Васютин А. М.
Преподаватель: Новицкая Ю. В.
Вариант: 4

Новосибирск, 2022 г.

ОГЛАВЛЕНИЕ

1	Лабораторная работа №1. Разработка классов, создание конструкторов и деструкторов	3
1.1	Цель работы	3
1.2	Задание на лабораторную работу	3
1.3	Решение	3
1.4	Исходный код	4
1.5	Выводы	8
2	Лабораторная работа №2. Переопределение операций	9
2.1	Цель работы	9
2.2	Задание на лабораторную работу	9
2.3	Решение	9
2.4	Исходный код	10
2.5	Выводы	18
3	Лабораторная работа №3. Организация ввода-вывода	19
3.1	Цель работы	19
3.2	Задание на лабораторную работу	19
3.3	Решение	19
3.4	Исходный код	20
3.5	Выводы	30
4	Лабораторная работа №4. Наследование	31
4.1	Цель работы	31
4.2	Задание на лабораторную работу	31
4.3	Решение	31
4.4	Исходный код	32
4.5	Выводы	45
5	Лабораторная работа №5. Создание динамического списка объектов, связанных наследованием	46
5.1	Цель работы	46
5.2	Задание на лабораторную работу	46
5.3	Решение	46
5.4	Исходный код	47
5.5	Выводы	63
6	Лабораторная работа №6. Обработка исключительных ситуаций	64
6.1	Цель работы	64
6.2	Задание на лабораторную работу	64
6.3	Решение	64

6.4 Исходный код	64
6.5 Выводы	81
Заключение	82

1. ЛАБОРАТОРНАЯ РАБОТА №1. РАЗРАБОТКА КЛАССОВ, СОЗДАНИЕ КОНСТРУКТОРОВ И ДЕКТРУКТОРОВ

1.1. Цель работы

Изучить структуру класса, механизм создания и использования, описание членов-данных класса и методов доступа к ним, возможность инициализации объектов класса с помощью конструкторов и уничтожение их с помощью деструкторов.

1.2. Задание на лабораторную работу

Реализовать класс в соответствии с вариантом. Класс должен обеспечивать набор методов для работы с данными. Создать перегруженные конструкторы: конструктор с параметрами, конструктор копирования, конструктор по умолчанию.

Реализовать указанные классы с динамическим выделением памяти для хранения некоторых полей. Создать деструктор для освобождения памяти. Посмотреть в отладчике, как вызываются конструкторы и деструкторы.

Составить демонстрационную программу. Для реализации демонстрационной программы использовать отдельный модуль. Программу построить с использованием проекта (см. файл «Пример проекта C++.docx»). Посмотреть работу программы в отладчике, обратить внимание на представление данных. Построить программу без отладочной информации. Обратить внимание на размер программы.

Построить класс для работы с квадратными матрицами. Класс должен включать соответствующие поля: порядок, набор коэффициентов (динамическое выделение памяти для набора коэффициентов).

Класс должен обеспечивать простейшие функции для работы с данными класса: транспонирование матрицы, вывод матрицы в удобной форме, сложение двух объектов класса.

1.3. Решение

Класс в языке C++ – это пользовательский тип, класс определяется ключевым словом *class* и сопутствующим синтаксисом. Класс может включать поля (статические и нестатические, включая битовые поля), методы (статические и нестатические), вложенные типы (классы, эnumерации, синонимы существующих типов), а так же шаблоны членов и шаблоны переменных (начиная с c++14).

У матрицы было создано 3 поля – размеры матрицы и указатель на массив коэффициентов матрицы, были реализованы конструкторы и деструкторы (листинги 2 и 3). Так же были реализованы методы *transpose*, *plus*, *fprint* и *print*.

```
~/c/nstu-labs * term_3/lab_1/build ./lab
Test 1:
[[0, 1, 2, 3, 4],
 [5, 6, 7, 8, 9],
 [10, 11, 12, 13, 14],
 [15, 16, 17, 18, 19],
 [20, 21, 22, 23, 24]]
Test 2: Транспонированная матрица
[[0, 5, 10, 15, 20],
 [1, 6, 11, 16, 21],
 [2, 7, 12, 17, 22],
 [3, 8, 13, 18, 23],
 [4, 9, 14, 19, 24]]
Test 3: заполнение матрицы
[[5, 5, 5, 5, 5],
 [5, 5, 5, 5, 5],
 [5, 5, 5, 5, 5],
 [5, 5, 5, 5, 5],
 [5, 5, 5, 5, 5]]
Test 4: диагональная матрица
[[5, 0, 0, 0, 0],
 [0, 5, 0, 0, 0],
 [0, 0, 5, 0, 0],
 [0, 0, 0, 5, 0],
 [0, 0, 0, 0, 5]]
Test 5: единичная матрица
[[1, 0, 0, 0, 0],
 [0, 1, 0, 0, 0],
 [0, 0, 1, 0, 0],
 [0, 0, 0, 1, 0],
 [0, 0, 0, 0, 1]]
Test 6: матрица нулей
[[0, 0, 0, 0, 0],
 [0, 0, 0, 0, 0],
 [0, 0, 0, 0, 0],
 [0, 0, 0, 0, 0],
 [0, 0, 0, 0, 0]]
Test 7: копирование и сложение
[[1, 3, 5, 7, 9],
 [11, 13, 15, 17, 19],
 [21, 23, 25, 27, 29],
 [31, 33, 35, 37, 39],
 [41, 43, 45, 47, 49]]
~/c/nstu-labs * term_3/lab_1/build
```

Рисунок 1 – Вывод работы программы.

1.4. Исходный код

Листинг 1 – main.cpp

```
#include "matrix.hpp"

#include <iostream>

int main() {
    std::cout << "Test 1:" << std::endl;
```

```

Matrix matrix(5);
for (int i = 0; i < 5 * 5; ++i) {
    matrix.set(i / 5, i % 5, i);
}
matrix.print();

std::cout << "Test 2: Транспонированная матрица" << std::endl;
Matrix tr = matrix.transpose();
tr.print();

std::cout << "Test 3: заполнение матрицы" << std::endl;
Matrix fill = Matrix::fill(5, 5);
fill.print();

std::cout << "Test 4: диагональная матрица" << std::endl;
Matrix diagonal = Matrix::diagonal(5, 5);
diagonal.print();
std::cout << "Test 5: единичная матрица" << std::endl;
Matrix identity = Matrix::identity(5);
identity.print();
std::cout << "Test 6: матрица нулей" << std::endl;
Matrix zeros = Matrix::zeros(5);
zeros.print();
std::cout << "Test 7: копирование и сложение" << std::endl;
Matrix copy(matrix);

matrix.plus(copy.plus(Matrix::fill(copy.getSize(), 1))).print();

return 0;
}

```

Листинг 2 – matrix.hpp

```

#include <cstdint>

#define MATRIX_DATATYPE double

using std::size_t;

class Matrix {
private:
    size_t size;
    MATRIX_DATATYPE *data;
    void init(size_t);

public:
    Matrix();
    Matrix(const size_t);
    Matrix(const Matrix &);
    ~Matrix();

```

```

Matrix plus(const Matrix &) const;
Matrix transpose() const;
void print() const;
MATRIX_DATATYPE get(size_t, size_t) const;
void set(size_t, size_t, MATRIX_DATATYPE);
size_t getSize() const;

static Matrix fill(size_t, MATRIX_DATATYPE);
static Matrix diagonal(size_t, MATRIX_DATATYPE);
static Matrix identity(size_t);
static Matrix zeros(size_t);
};

```

Листинг 3 – matrix.cpp

```

#include "matrix.hpp"

#include <cmath>
#include <cstdint>
#include <iomanip>
#include <iostream>
#include <limits>
#include <ostream>

void Matrix::init(size_t size) {
    this->size = size;
    size_t s = this->size * this->size;
    if (size > 0)
        this->data = new MATRIX_DATATYPE[s];
}

Matrix::Matrix() { this->init(0); }

Matrix::Matrix(const size_t size) { this->init(size); }

Matrix::Matrix(const Matrix &matrix) {
    size_t size = matrix.getSize();
    this->init(size);
    for (size_t i = 0; i < size * size; ++i) {
        set(i / size, i % size, matrix.get(i / size, i % size));
    }
}

Matrix::~Matrix() {
    if (size > 0)
        delete[] this->data;
}

Matrix Matrix::plus(const Matrix &other) const {
    Matrix matrix(this->size);
    for (size_t i = 0; i < this->size; ++i) {

```

```

        for (size_t j = 0; j < this->size; ++j) {
            matrix.set(i, j, this->get(i, j) + other.get(i, j));
        }
    }
    return matrix;
}

```

```

Matrix Matrix::transpose() const { // TODO: rewrite
    Matrix result(this->size);

```

```

        for (size_t i = 0; i < result.getSize(); ++i) {
            for (size_t j = i; j < result.getSize(); ++j) {
                MATRIX_DATATYPE tmp = this->get(i, j);
                result.set(i, j, this->get(j, i));
                result.set(j, i, tmp);
            }
        }
    }

```

```

    // for (size_t i = 0; i < this->size; ++i) {
    //     for (size_t j = 0; j < this->size; ++j) {
    //         result.set(i, j, this->get(j, i));
    //     }
    // }
    // }
    return result;
}

```

```

void Matrix::print() const {
    std::cout << "[";
    for (size_t i = 0; i < this->size; ++i) {
        if (i != 0) {
            std::cout << " ";
        }
        std::cout << "[";
        for (size_t j = 0; j < this->size; ++j) {
            std::cout << this->get(i, j);
            if (j + 1 < this->size) {
                std::cout << ", ";
            }
        }
        std::cout << "]";
        if (i + 1 < this->size) {
            std::cout << ",";
            std::cout << std::endl;
        }
    }
    std::cout << "]" << std::endl;
}

```

```

MATRIX_DATATYPE Matrix::get(size_t x, size_t y) const {
    return this->data[x * this->size + y];
}

```

```

void Matrix::set(size_t x, size_t y, MATRIX_DATATYPE v) {
    this->data[x * this->size + y] = v;
}

```



```

}

size_t Matrix::getSize() const { return this->size; }

Matrix Matrix::fill(size_t size, MATRIX_DATATYPE value) {
    Matrix result(size);
    size_t s = result.getSize();
    for (size_t i = 0; i < s * s; ++i) {
        result.set(i / s, i % s, value);
    }
    return result;
}

Matrix Matrix::diagonal(size_t size, MATRIX_DATATYPE value) {
    Matrix result = Matrix::zeros(size);
    for (size_t i = 0; i < result.getSize(); ++i) {
        result.set(i, i, value);
    }
    return result;
}

Matrix Matrix::identity(size_t size) { return Matrix::diagonal(size, 1); }

Matrix Matrix::zeros(size_t size) { return Matrix::fill(size, 0.0); }

```

1.5. Выводы

В ходе лабораторной работы были изучены базовые конструкции языка, связанные с классами. Были реализованы методы создания и разрушения объектов, а так же искомые методы. В реализации работы был применен один из ключевых механизмов объектно-ориентированного программирования – инкапсуляция, объединение данных и функций (методов), работающих с ними. Так же лабораторная работа состоит из нескольких файлов, следовательно, были отработаны навыки работы с многофайловыми проектами на языке C++.

2. ЛАБОРАТОРНАЯ РАБОТА №2. ПЕРЕОПРЕДЕЛЕНИЕ ОПЕРАЦИЙ

2.1. Цель работы

Ознакомиться с особенностями использования дружественных классов и функций, а также возможностью получения законченного нового типа данных, определив для него допустимые операции с помощью перегрузки операторов. Изучить структуру класса, механизм создания и использования, описание членов-данных класса и методов доступа к ним, возможность инициализации объектов класса с помощью конструкторов и уничтожение их с помощью деструкторов.

2.2. Задание на лабораторную работу

Для разработанного класса из лабораторной работы №1 реализовать набор операций для работы с объектами класса: сложение (как метод класса), вычитание (как дружественную функцию), присваивание (как метод класса), инкремент постфиксный и инкремент префиксный (как методы класса) (разобраться и вникнуть, в чем между ними разница!), приведение к некоторому типу (как метод класса).

Дополнить демонстрационную программу, продемонстрировав все перегруженные операции.

2.3. Решение

Механизмы перегрузки операторов с языке с++ позволяют пользовательским типам мимикрировать под встроенные типы или же расширять функционал пользовательских типов используя привычный синтаксис его использования.

Для класса были реализованы основные арифметические операции, а также оператор `()` для двух аргументов, что позволяет создать удобный интерфейс получения значения или ссылки определенного коэффициента матрицы. Префиксный и постфиксный инкременты и декременты синтаксически отличаются наличием у постфиксных операторов аргумента. Префиксный оператор сначала проводит инкрементирование объекта и возвращает его, а постфиксный оператор создает копию объекта, инкрементирует его и возвращает копию, следовательно, операции производимые с объектом, который был вернут постфиксным инкрементом не изменяют исходный объект, т.к. он является отдельным объектом.

```

Test 8: Префиксный инкремент
[[2, 2, 2, 2, 2],
 [2, 2, 2, 2, 2],
 [2, 2, 2, 2, 2],
 [2, 2, 2, 2, 2],
 [2, 2, 2, 2, 2]]
[[2, 2, 2, 2, 2],
 [2, 2, 2, 2, 2],
 [2, 2, 2, 2, 2],
 [2, 2, 2, 2, 2],
 [2, 2, 2, 2, 2]]
Test 9: Постфиксный инкремент
[[1, 1, 1, 1, 1],
 [1, 1, 1, 1, 1],
 [1, 1, 1, 1, 1],
 [1, 1, 1, 1, 1],
 [1, 1, 1, 1, 1]]
[[2, 2, 2, 2, 2],
 [2, 2, 2, 2, 2],
 [2, 2, 2, 2, 2],
 [2, 2, 2, 2, 2],
 [2, 2, 2, 2, 2]]
Test 10: Присваивание
[[1, 1, 1, 1, 1],
 [1, 1, 1, 1, 1],
 [1, 1, 1, 1, 1],
 [1, 1, 1, 1, 1],
 [1, 1, 1, 1, 1]]
[[1, 0, 0, 0, 0],
 [0, 1, 0, 0, 0],
 [0, 0, 1, 0, 0],
 [0, 0, 0, 1, 0],
 [0, 0, 0, 0, 1]]
Test 11: operator+
[[2, 1, 1, 1, 1],
 [1, 2, 1, 1, 1],
 [1, 1, 2, 1, 1],
 [1, 1, 1, 2, 1],
 [1, 1, 1, 1, 2]]
Test 12: operator-
[[0, 1, 1, 1, 1],
 [1, 0, 1, 1, 1],
 [1, 1, 0, 1, 1],
 [1, 1, 1, 0, 1],
 [1, 1, 1, 1, 0]]
Test 13: to size_t
5
~/c/nstu-labs * term_3/lab_2/build |

```

Рисунок 1 – Вывод работы программы.

2.4. Исходный код

Листинг 4 – main.cpp

```

#include "matrix.hpp"

#include <iostream>

```

```

int main() {
    std::cout << "Test 1:" << std::endl;
    Matrix matrix(5);
    for (int i = 0; i < 5 * 5; ++i) {
        matrix(i / 5, i % 5) = i;
    }
    matrix.print();

    std::cout << "Test 2: Транспонированная матрица" << std::endl;
    Matrix tr = matrix.transpose();
    tr.print();

    std::cout << "Test 3: заполнение матрицы" << std::endl;
    Matrix fill = Matrix::fill(5, 5);
    fill.print();

    std::cout << "Test 4: диагональная матрица" << std::endl;
    Matrix diagonal = Matrix::diagonal(5, 5);
    diagonal.print();
    std::cout << "Test 5: единичная матрица" << std::endl;
    Matrix identity = Matrix::identity(5);
    identity.print();
    std::cout << "Test 6: матрица нулей" << std::endl;
    Matrix zeros = Matrix::zeros(5);
    zeros.print();
    std::cout << "Test 7: копирование и сложение" << std::endl;
    Matrix copy(matrix);
    matrix.plus(copy.plus(Matrix::fill(copy.getSize(), 1))).print();

    std::cout << "Test 8: Префиксный инкремент" << std::endl;
    Matrix a = Matrix::fill(5, 1);
    (++a).print();
    a.print();
    std::cout << "Test 9: Постфиксный инкремент" << std::endl;
    Matrix b = Matrix::fill(5, 1);
    (b++).print();
    b.print();

    std::cout << "Test 10: Присваивание" << std::endl;
    Matrix c = Matrix::fill(5, 1);
    Matrix d = Matrix::identity(5);

    Matrix e;
    e = c;
    e = d;

    std::cout << "Test 11: operator+" << std::endl;
    (c + d).print();

    std::cout << "Test 12: operator-" << std::endl;
    (c - d).print();

    std::cout << "Test 13: to size_t" << std::endl;

```

```

std::cout << size_t(Matrix(5)) << std::endl;

return 0;
}

```

Листинг 5 – matrix.hpp

```

#include <cstdint>

#define MATRIX_DATATYPE double

using std::size_t;

class Matrix {
private:
    size_t size;
    MATRIX_DATATYPE *data;
    void init(size_t);

    MATRIX_DATATYPE max() const;
    MATRIX_DATATYPE min() const;

    bool validateIndexes(size_t, size_t) const;

public:
    Matrix();
    Matrix(const size_t);
    Matrix(const Matrix &);
    ~Matrix();

    Matrix plus(const Matrix &) const;
    Matrix transpose() const;
    void print() const;
    // MATRIX_DATATYPE get(size_t, size_t) const;
    // void set(size_t, size_t, MATRIX_DATATYPE);
    size_t getSize() const;

    static Matrix fill(size_t, MATRIX_DATATYPE);
    static Matrix diagonal(size_t, MATRIX_DATATYPE);
    static Matrix identity(size_t);
    static Matrix zeros(size_t);

    Matrix operator+() const;
    Matrix operator+(const Matrix &) const;
    double operator()(size_t, size_t) const;
    MATRIX_DATATYPE &operator()(size_t, size_t);
    Matrix &operator++();
    Matrix operator++(int);
    Matrix &operator--();
    Matrix operator--(int);

```

```

    operator size_t();

    Matrix &operator=(const Matrix &other) noexcept;
};

const Matrix operator-(const Matrix &);
const Matrix operator-(const Matrix &, const Matrix &);
const Matrix operator*(MATRIX_DATATYPE, const Matrix &);
const Matrix operator*(const Matrix &, MATRIX_DATATYPE);
const Matrix operator*(const Matrix &, const Matrix &);

```

Листинг 6 – matrix.cpp

```

#include "matrix.hpp"

#include <cmath>
#include <cstdint>
#include <iomanip>
#include <iostream>

void Matrix::init(size_t size) {
    this->size = size;
    size_t s = this->size * this->size;
    this->data = new MATRIX_DATATYPE[s];
}

Matrix::Matrix() {
    this->init(0);
}

Matrix::Matrix(const size_t size) {
    this->init(size);
}

Matrix::Matrix(const Matrix &matrix) {
    size_t size = matrix.getSize();
    this->init(size);
    for (size_t i = 0; i < size * size; ++i) {
        operator()(i / size, i % size) = matrix(i / size, i % size);
    }
}

Matrix::~Matrix() {
    delete[] this->data;
}

Matrix Matrix::plus(const Matrix &other) const {
    Matrix matrix(this->size);
    for (size_t i = 0; i < this->size; ++i) {
        for (size_t j = 0; j < this->size; ++j) {

```

```

        matrix(i, j) = operator()(i, j) + other(i, j);
    }
}
return matrix;
}

Matrix Matrix::transpose() const {
    size_t size = getSize();

    Matrix result(size);

    for (size_t i = 0; i < size; ++i) {
        for (size_t j = i; j < size; ++j) {
            MATRIX_DATATYPE tmp = operator()(i, j);
            result(i, j) = operator()(j, i);
            result(j, i) = tmp;
        }
    }

    // for (size_t i = 0; i < this->size; ++i) {
    //     for (size_t j = 0; j < this->size; ++j) {
    //         result.set(i, j, this->get(j, i));
    //     }
    // }
    return result;
}

void Matrix::print() const {
    size_t size = getSize();
    std::cout << "[";
    for (size_t i = 0; i < size; ++i) {
        if (i != 0) {
            std::cout << " ";
        }
        std::cout << "[";
        for (size_t j = 0; j < size; ++j) {
            std::cout << operator()(i, j);
            if (j + 1 < size) {
                std::cout << ", ";
            }
        }
        std::cout << "];";
        if (i + 1 < size) {
            std::cout << ", ";
            std::cout << std::endl;
        }
    }
    std::cout << "]" << std::endl;
}

// MATRIX_DATATYPE Matrix::get(size_t x, size_t y) const {
//     return this->data[x * this->size + y];
// }

```

```

// void Matrix::set(size_t x, size_t y, MATRIX_DATATYPE v) {
//     this->data[x * this->size + y] = v;
// }

size_t Matrix::getSize() const {
    return this->size;
}

MATRIX_DATATYPE Matrix::max() const {
    int max = operator()(0, 0);
    size_t size = getSize();

    for (size_t i = 1; i < size * size; ++i) {
        int el = operator()(i / size, i % size);
        if (el > max) {
            max = el;
        }
    }
    return max;
}

MATRIX_DATATYPE Matrix::min() const {
    int min = operator()(0, 0);
    size_t size = getSize();

    for (size_t i = 1; i < size * size; ++i) {
        int el = operator()(i / size, i % size);
        if (el < min) {
            min = el;
        }
    }
    return min;
}

Matrix Matrix::fill(size_t size, MATRIX_DATATYPE value) {
    Matrix result(size);
    size_t s = result.getSize();
    for (size_t i = 0; i < s * s; ++i) {
        result(i / s, i % s) = value;
    }
    return result;
}

Matrix Matrix::diagonal(size_t size, MATRIX_DATATYPE value) {
    Matrix result = Matrix::zeros(size);
    for (size_t i = 0; i < result.getSize(); ++i) {
        result(i, i) = value;
    }
    return result;
}

Matrix Matrix::identity(size_t size) {
    return Matrix::diagonal(size, 1);
}

```



```

}

Matrix Matrix::zeros(size_t size) {
    return Matrix::fill(size, 0.0);
}

Matrix Matrix::operator+() const {
    return Matrix(*this);
}

Matrix Matrix::operator+(const Matrix &other) const {
    size_t size = getSize();

    Matrix result(*this);

    for (size_t i = 0; i < size * size; ++i) {
        result(i / size, i % size) += other(i / size, i % size);
    }

    return result;
}

Matrix &Matrix::operator++() {
    size_t size = getSize();
    size_t s = size * size;
    for (size_t i = 0; i < s; ++i) {
        operator()(i / size, i % size) = operator()(i / size, i % size) + 1;
    }
    return *this;
}

Matrix Matrix::operator++(int) {
    Matrix result(*this);
    operator++();
    return result;
}

Matrix &Matrix::operator--() {
    size_t size = getSize();
    size_t s = size * size;
    for (size_t i = 0; i < s; ++i) {
        operator()(i / size, i % size) = operator()(i / size, i % size) + 1;
    }
    return *this;
}

Matrix Matrix::operator--(int) {
    Matrix result(*this);
    operator--();
    return result;
}

Matrix::operator size_t() {
    return getSize();
}

```

```

}

Matrix &Matrix::operator=(const Matrix &other) noexcept {
    if (&other == this) {
        return *this;
    }
    size = other.getSize();
    delete[] data;
    this->data = new MATRIX_DATATYPE[size * size];
    for (size_t i = 0; i < size * size; ++i) {
        operator()(i / size, i % size) = other(i / size, i % size);
    }

    print();

    return *this;
}

MATRIX_DATATYPE Matrix::operator()(size_t row, size_t col) const {
    return this->data[row * getSize() + col];
}

MATRIX_DATATYPE &Matrix::operator()(size_t row, size_t col) {
    return this->data[row * getSize() + col];
}

const Matrix operator-(const Matrix &matrix) {
    return -1.0 * matrix;
}

const Matrix operator-(const Matrix &matrix, const Matrix &other) {
    const Matrix o = -other;
    return matrix + o;
}

const Matrix operator*(MATRIX_DATATYPE value, const Matrix &matrix) {
    size_t size = matrix.getSize();
    Matrix result(size);
    for (size_t i = 0; i < size; ++i) {
        for (size_t j = 0; j < size; ++j) {
            result(i, j) = matrix(i, j) * value;
        }
    }
    return result;
}

const Matrix operator*(const Matrix &matrix, MATRIX_DATATYPE value) {
    return value * matrix;
}

const Matrix operator*(const Matrix &matrix, const Matrix &other) {
    // TODO:
    return matrix;
}

```

2.5. Выводы

В ходе лабораторной работы были изучены механизмы перегрузки операторов, что позволяет переопределить поведение привычных операций для пользовательских типов, что позволяет увеличить читаемость кода в случае соответствия семантик оператора и переопределения, например, конкатенация строк.

3. ЛАБОРАТОРНАЯ РАБОТА №3. ОРГАНИЗАЦИЯ ВВОДА-ВЫВОДА

3.1. Цель работы

Изучить работу потоков ввода-вывода и реализацию перегрузки потоков ввода-вывода на стандартные устройства и в файл для разработанных классов.

3.2. Задание на лабораторную работу

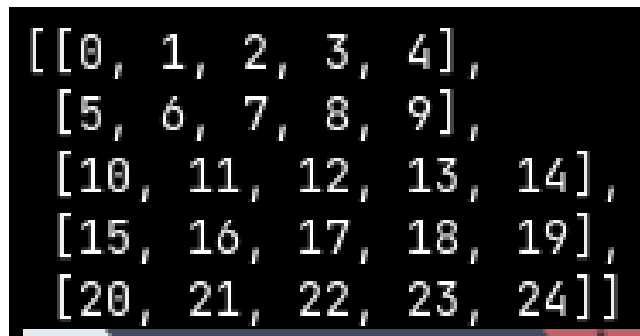
Для класса из лабораторной работы №2 перегрузить операции ввода/вывода, позволяющие осуществлять ввод и вывод в удобной форме объектов классов:

- вывод объекта класса в текстовый файл;
- вывод объекта класса в двоичный файл;
- ввод объекта класса из двоичного файла.

Дополнить демонстрационную программу, продемонстрировав все перегруженные операции.

3.3. Решение

Для работы с файлами был создан класс `File`, который является фасадом для объекта `std::fstream`, а так же реализованы перегрузки оператора левого и правого сдвига (« и »)



```
[[0, 1, 2, 3, 4],  
 [5, 6, 7, 8, 9],  
 [10, 11, 12, 13, 14],  
 [15, 16, 17, 18, 19],  
 [20, 21, 22, 23, 24]]
```

Рисунок 1 – Вывод работы программы.

```
% ~ /c/nstu-labs  * term_3/lab_3 cat ./assets/out.txt
[[0, 1, 2, 3, 4],
 [5, 6, 7, 8, 9],
 [10, 11, 12, 13, 14],
 [15, 16, 17, 18, 19],
 [20, 21, 22, 23, 24]]

% ~ /c/nstu-labs  * term_3/lab_3 xxd ./assets/out.b
00000000: 0500 0000 0000 0000 0000 0000 0000 0000 .....
00000010: 0000 0000 0000 f03f 0000 0000 0000 0040 .....?.....@
00000020: 0000 0000 0000 0840 0000 0000 0000 1040 .....@.....@
00000030: 0000 0000 0000 1440 0000 0000 0000 1840 .....@.....@
00000040: 0000 0000 0000 1c40 0000 0000 0000 2040 .....@.....@
00000050: 0000 0000 0000 2240 0000 0000 0000 2440 ....."@.....$@
00000060: 0000 0000 0000 2640 0000 0000 0000 2840 .....&@.....(@
00000070: 0000 0000 0000 2a40 0000 0000 0000 2c40 .....*@.....,@
00000080: 0000 0000 0000 2e40 0000 0000 0000 3040 .....@.....0@
00000090: 0000 0000 0000 3140 0000 0000 0000 3240 .....1@.....2@
000000a0: 0000 0000 0000 3340 0000 0000 0000 3440 .....3@.....4@
000000b0: 0000 0000 0000 3540 0000 0000 0000 3640 .....5@.....6@
000000c0: 0000 0000 0000 3740 0000 0000 0000 3840 .....7@.....8@

% ~ /c/nstu-labs  * term_3/lab_3 xxd ./assets/out.txt
00000000: 5b5b 302c 2031 2c20 322c 2033 2c20 345d [[0, 1, 2, 3, 4]
00000010: 2c0a 205b 352c 2036 2c20 372c 2038 2c20 ,. [5, 6, 7, 8,
00000020: 395d 2c0a 205b 3130 2c20 3131 2c20 3132 9],. [10, 11, 12
00000030: 2c20 3133 2c20 3134 5d2c 0a20 5b31 352c , 13, 14],. [15,
00000040: 2031 362c 2031 372c 2031 382c 2031 395d 16, 17, 18, 19]
00000050: 2c0a 205b 3230 2c20 3231 2c20 3232 2c20 ,. [20, 21, 22,
00000060: 3233 2c20 3234 5d5d 0a 23, 24]].

% ~ /c/nstu-labs  * term_3/lab_3
```

Рисунок 2 – Полученные файлы (*in.b* – копия *out.b*).

3.4. Исходный код

Листинг 7 – main.cpp

```
#include "matrix.hpp"

#include <cstdint>
#include <fstream>
#include <ios>
#include <iostream>
#include <ostream>

int main() {
    std::cout << "Test 1:" << std::endl;
    Matrix matrix(5);
    for (int i = 0; i < 5 * 5; ++i) {
        matrix(i / 5, i % 5) = i;
    }
}
```

```

matrix.print();

std::cout << "Test 2: Транспонированная матрица" << std::endl;
Matrix tr = matrix.transpose();
tr.print();

std::cout << "Test 3: заполнение матрицы" << std::endl;
Matrix fill = Matrix::fill(5, 5);
fill.print();

std::cout << "Test 4: диагональная матрица" << std::endl;
Matrix diagonal = Matrix::diagonal(5, 5);
diagonal.print();
std::cout << "Test 5: единичная матрица" << std::endl;
Matrix identity = Matrix::identity(5);
identity.print();
std::cout << "Test 6: матрица нулей" << std::endl;
Matrix zeros = Matrix::zeros(5);
zeros.print();
std::cout << "Test 7: копирование и сложение" << std::endl;
Matrix copy(matrix);
matrix.plus(copy.plus(Matrix::fill(copy.getSize(), 1))).print();

std::cout << "Test 8: Префиксный инкремент" << std::endl;
Matrix a = Matrix::fill(5, 1);
(++a).print();
a.print();
std::cout << "Test 9: Постфиксный инкремент" << std::endl;
Matrix b = Matrix::fill(5, 1);
(b++).print();
b.print();

std::cout << "Test 10: Присваивание" << std::endl;
Matrix c = Matrix::fill(5, 1);
Matrix d = Matrix::identity(5);

Matrix e;
e = c;
e = d;

std::cout << "Test 11: operator+" << std::endl;
(c + d).print();

std::cout << "Test 12: operator-" << std::endl;
(c - d).print();

std::cout << "Test 13: to size_t" << std::endl;
std::cout << size_t(matrix) << std::endl;

File off("./assets/out.txt", std::ios_base::out);
off << matrix;
off.close();

File obf("./assets/out.b", std::ios_base::binary | std::ios_base::out);

```

```

    obf << matrix;
    obf.close();

    Matrix mmm;
    File ibf("./assets/in.b", std::ios_base::binary | std::ios_base::in);
    ibf >> mmm;
    ibf.close();
    mmm.print();

    return 0;
}

```

Листинг 8 – matrix.hpp

```

#include <cstdlib>
#include <fstream>
#include <ostream>

#include "file.hpp"

#define MATRIX_DATATYPE double

using std::size_t;

class Matrix {
private:
    size_t size;
    MATRIX_DATATYPE *data;
    void init(size_t);

    MATRIX_DATATYPE max() const;
    MATRIX_DATATYPE min() const;

    bool validateIndexes(size_t, size_t) const;

public:
    Matrix();
    Matrix(const size_t);
    Matrix(const Matrix &);
    ~Matrix();

    Matrix plus(const Matrix &) const;
    Matrix transpose() const;
    void print() const;
    // MATRIX_DATATYPE get(size_t, size_t) const;
    // void set(size_t, size_t, MATRIX_DATATYPE);
    size_t getSize() const;

    static Matrix fill(size_t, MATRIX_DATATYPE);
    static Matrix diagonal(size_t, MATRIX_DATATYPE);

```

```

static Matrix identity(size_t);
static Matrix zeros(size_t);

Matrix operator+() const;
Matrix operator+(const Matrix &) const;
MATRIX_DATATYPE operator()(size_t, size_t) const;
MATRIX_DATATYPE &operator()(size_t, size_t);
Matrix &operator++();
Matrix operator++(int);
Matrix &operator--();
Matrix operator--(int);

operator size_t();

Matrix &operator=(const Matrix &other) noexcept;
};

const Matrix operator-(const Matrix &);
const Matrix operator-(const Matrix &, const Matrix &);
const Matrix operator*(MATRIX_DATATYPE, const Matrix &);
const Matrix operator*(const Matrix &, MATRIX_DATATYPE);
const Matrix operator*(const Matrix &, const Matrix &);

File &operator>>(File &, Matrix &);
File &operator<<(File &, const Matrix &);

```

Листинг 9 – matrix.cpp

```

#include "matrix.hpp"

#include <cmath>
#include <cstdint>
#include <cstdint>
#include <cstdlib>
#include <iomanip>
#include <ios>
#include <iostream>

void Matrix::init(size_t size) {
    this->size = size;
    size_t s = this->size * this->size;
    // if (size > 0)
    this->data = new MATRIX_DATATYPE[s];
}

Matrix::Matrix() {
    this->init(0);
}

Matrix::Matrix(const size_t size) {
    this->init(size);
}

```



```

}

Matrix::Matrix(const Matrix &matrix) {
    size_t size = matrix.getSize();
    this->init(size);
    for (size_t i = 0; i < size * size; ++i) {
        operator()(i / size, i % size) = matrix(i / size, i % size);
    }
}

Matrix::~Matrix() {
    // if (size > 0)
    delete[] this->data;
}

Matrix Matrix::plus(const Matrix &other) const {
    Matrix matrix(this->size);
    for (size_t i = 0; i < this->size; ++i) {
        for (size_t j = 0; j < this->size; ++j) {
            matrix(i, j) = operator()(i, j) + other(i, j);
        }
    }
    return matrix;
}

Matrix Matrix::transpose() const {
    size_t size = getSize();

    Matrix result(size);

    for (size_t i = 0; i < size; ++i) {
        for (size_t j = i; j < size; ++j) {
            MATRIX_DATATYPE tmp = operator()(i, j);
            result(i, j) = operator()(j, i);
            result(j, i) = tmp;
        }
    }

    // for (size_t i = 0; i < this->size; ++i) {
    //     for (size_t j = 0; j < this->size; ++j) {
    //         result.set(i, j, this->get(j, i));
    //     }
    // }
    return result;
}

void Matrix::print() const {
    size_t size = getSize();
    std::cout << "[";
    for (size_t i = 0; i < size; ++i) {
        if (i != 0) {
            std::cout << " ";
        }
    }
}

```

```

std::cout << "[";
for (size_t j = 0; j < size; ++j) {
    std::cout << operator()(i, j);
    if (j + 1 < size) {
        std::cout << ", ";
    }
}
std::cout << "]";
if (i + 1 < size) {
    std::cout << ",";
    std::cout << std::endl;
}
}
std::cout << "]" << std::endl;
}

// MATRIX_DATATYPE Matrix::get(size_t x, size_t y) const {
//     return this->data[x * this->size + y];
// }

// void Matrix::set(size_t x, size_t y, MATRIX_DATATYPE v) {
//     this->data[x * this->size + y] = v;
// }

size_t Matrix::getSize() const {
    return this->size;
}

MATRIX_DATATYPE Matrix::max() const {
    int max = operator()(0, 0);
    size_t size = getSize();

    for (size_t i = 1; i < size * size; ++i) {
        int el = operator()(i / size, i % size);
        if (el > max) {
            max = el;
        }
    }
    return max;
}

MATRIX_DATATYPE Matrix::min() const {
    int min = operator()(0, 0);
    size_t size = getSize();

    for (size_t i = 1; i < size * size; ++i) {
        int el = operator()(i / size, i % size);
        if (el < min) {
            min = el;
        }
    }
    return min;
}

```

```

Matrix Matrix::fill(size_t size, MATRIX_DATATYPE value) {
    Matrix result(size);
    size_t s = result.getSize();
    for (size_t i = 0; i < s * s; ++i) {
        result(i / s, i % s) = value;
    }
    return result;
}

Matrix Matrix::diagonal(size_t size, MATRIX_DATATYPE value) {
    Matrix result = Matrix::zeros(size);
    for (size_t i = 0; i < result.getSize(); ++i) {
        result(i, i) = value;
    }
    return result;
}

Matrix Matrix::identity(size_t size) {
    return Matrix::diagonal(size, 1);
}

Matrix Matrix::zeros(size_t size) {
    return Matrix::fill(size, 0.0);
}

Matrix Matrix::operator+() const {
    return Matrix(*this);
}

Matrix Matrix::operator+(const Matrix &other) const {
    size_t size = getSize();

    Matrix result(*this);

    for (size_t i = 0; i < size * size; ++i) {
        result(i / size, i % size) += other(i / size, i % size);
    }

    return result;
}

Matrix &Matrix::operator++() {
    size_t size = getSize();
    size_t s = size * size;
    for (size_t i = 0; i < s; ++i) {
        operator()(i / size, i % size) = operator()(i / size, i % size) + 1;
    }
    return *this;
}

Matrix Matrix::operator++(int) {
    Matrix result(*this);
    operator++();
    return result;
}

```

```

}

Matrix &Matrix::operator--() {
    size_t size = getSize();
    size_t s = size * size;
    for (size_t i = 0; i < s; ++i) {
        operator()(i / size, i % size) = operator()(i / size, i % size) + 1;
    }
    return *this;
}

Matrix Matrix::operator--(int) {
    Matrix result(*this);
    operator--();
    return result;
}

Matrix::operator size_t() {
    return getSize();
}

Matrix &Matrix::operator=(const Matrix &other) noexcept {
    if (&other == this) {
        return *this;
    }
    size = other.getSize();
    delete[] data;
    this->data = new MATRIX_DATATYPE[size * size];
    for (size_t i = 0; i < size * size; ++i) {
        operator()(i / size, i % size) = other(i / size, i % size);
    }

    return *this;
}

MATRIX_DATATYPE Matrix::operator()(size_t row, size_t col) const {
    return this->data[row * getSize() + col];
}

MATRIX_DATATYPE &Matrix::operator()(size_t row, size_t col) {
    return this->data[row * getSize() + col];
}

const Matrix operator-(const Matrix &matrix) {
    return -1.0 * matrix;
}

const Matrix operator-(const Matrix &matrix, const Matrix &other) {
    const Matrix o = -other;
    return matrix + o;
}

const Matrix operator*(MATRIX_DATATYPE value, const Matrix &matrix) {
    size_t size = matrix.getSize();

```

```

Matrix result(size);
for (size_t i = 0; i < size; ++i) {
    for (size_t j = 0; j < size; ++j) {
        result(i, j) = matrix(i, j) * value;
    }
}
return result;
}

const Matrix operator*(const Matrix &matrix, MATRIX_DATATYPE value) {
    return value * matrix;
}

const Matrix operator*(const Matrix &matrix, const Matrix &other) {
    // TODO:
    return matrix;
}

File &operator>>(File &file, Matrix &matrix) {
    if ((file.getMode() & std::ios_base::in) <= 0) {
        return file;
    }
    std::fstream &fs = file.getFStream();
    if ((file.getMode() & std::ios_base::binary) > 0) {
        size_t size;
        MATRIX_DATATYPE el;

        fs.read((char *)&size, sizeof(size_t));
        std::cout << size << std::endl;
        matrix = Matrix::zeros(size);

        for (size_t i = 0; i < size * size; ++i) {
            fs.read((char *)&el, sizeof(MATRIX_DATATYPE));
            matrix(i / size, i % size) = el;
        }
    } else {
        // TODO
    }
    return file;
}

File &operator<<(File &file, const Matrix &matrix) {
    if ((file.getMode() & std::ios_base::out) <= 0) {
        return file;
    }

    size_t size = matrix.getSize();
    std::fstream &fs = file.getFStream();

    if ((file.getMode() & std::ios_base::binary) > 0) {
        fs.write((char *)&size, sizeof(size_t));
        for (size_t i = 0; i < size * size; ++i) {
            MATRIX_DATATYPE el = matrix(i / size, i % size);
            fs.write((char *)&el, sizeof(MATRIX_DATATYPE));
        }
    }
}

```

```

    }
} else {
    fs << "[";
    for (size_t i = 0; i < size; ++i) {
        if (i != 0) {
            fs << " ";
        }
        fs << "[";
        for (size_t j = 0; j < size; ++j) {
            fs << matrix(i, j);
            if (j + 1 < size) {
                fs << ", ";
            }
        }
        fs << "]\n";
        if (i + 1 < size) {
            fs << ",\n";
        }
    }
    fs << "]\n";
}
return file;
}

```

Листинг 10 – file.hpp

```

#include <fstream>
#include <ios>

class File {
private:
    int _mode;
    std::fstream _fs;

public:
    File(const char *, std::ios_base::openmode);
    ~File();
    void close();
    int getMode();
    std::fstream &getFStream();
};

```

Листинг 11 – file.cpp

```

#include "file.hpp"

#include <ios>

```

```

File::File(const char *filename, std::ios_base::openmode mode) : _mode(mode) {
    _fs.open(filename, mode);
}

File::~~File() {
    close();
}

void File::close() {
    _fs.close();
}

int File::getMode() {
    return static_cast<int>(_mode);
}

std::fstream &File::getFStream() {
    return _fs;
}

```

3.5. Выводы

В ходе лабораторной работы были изучены механизмы работы с файлами в языке с++, что является важной частью работы с данными.

4. ЛАБОРАТОРНАЯ РАБОТА №4. НАСЛЕДОВАНИЕ

4.1. Цель работы

Изучить механизм наследования и возможности порождения новых типов данных на основе уже существующих классов, изучить определение виртуальных функций и их использование для позднего связывания.

4.2. Задание на лабораторную работу

Для классов предыдущей лабораторной работы реализовать иерархию, изменяя отдельные методы и добавляя члены-данные (по усмотрению студента и преподавателя). В иерархию должно входить 2 производных класса. Один из методов должен быть виртуальным.

Дополнить демонстрационную программу так, чтобы она демонстрировала создание, копирование объектов родственных типов, работу виртуальных функций.

4.3. Решение

Наследование – один из механизмов объектно-ориентированного программирования, позволяющих создавая на основе каких либо классов производные, что убирает потребность в повторе кода и облегчает поддержку кода. В этой лабораторной работе была переработана структура классов, был создан базовый класс, представляющий матрицу, и имеющий 3 наследника – квадратную матрицу и 2 вектора (вертикальный и горизонтальный).

Также был реплизован механизм виртуальных функций на основе метода *fprint*. Таким образом, был реализован еще один механизм объектно-ориентированного программирования – полиморфизм. Объекты типов с виртуальными функциями могут вызывая одни и те же функции получать их разное поведение (если вызывать их у указателя или ссылки на объект базового класса, но не значения), что показано в лабораторной работе (вектор печатается иначе).


```

Test 17: Overload vectors print
[5, 5, 5, 5, 5, 5, 5, 5, 5, 5]
[5, 5, 5, 5, 5, 5, 5, 5, 5, 5]
Test 18: VectorH to VectorV
[5, 5, 5, 5, 5, 5, 5, 5, 5, 5]
[5,
 5,
 5,
 5,
 5,
 5,
 5,
 5,
 5,
 5]

```

Рисунок 1 – Вывод работы программы.

4.4. Исходный код

Листинг 12 – main.cpp

```

#include "matrix.hpp"

#include <cstdint>
#include <fstream>
#include <ios>
#include <iostream>
#include <ostream>

static int test_num = 0;

#define TESTCASE_BEGIN(TEXT)
    test_num++;
    std::cout << "Test " << test_num << ": " TEXT << std::endl;
    {
#define TESTCASE_END() }

int main() {
    TESTCASE_BEGIN("");
    SquareMatrix m(5);
    for (int i = 0; i < 5 * 5; ++i) {
        m(i / 5, i % 5) = i;
    }
    m.print();
    TESTCASE_END();

    TESTCASE_BEGIN("Транспонированная матрица");

```

```

SquareMatrix m(5);
for (int i = 0; i < 5 * 5; ++i) {
    m(i / 5, i % 5) = i;
}
m.print();
SquareMatrix tr = m.transpose();
tr.print();
TESTCASE_END();

TESTCASE_BEGIN("Заполнение матрицы");
Matrix fill = Matrix::fill(5, 5, 5);
fill.print();
SquareMatrix sqfill = SquareMatrix::fill(5, 5);
sqfill.print();
TESTCASE_END();

TESTCASE_BEGIN("Диагональная матрица");
SquareMatrix diagonal = SquareMatrix::diagonal(5, 5);
diagonal.print();
TESTCASE_END();

TESTCASE_BEGIN("Единичная матрица");
SquareMatrix identity = SquareMatrix::identity(5);
identity.print();
TESTCASE_END();

TESTCASE_BEGIN("Матрица нулей");
SquareMatrix zeros = Matrix::zeros(5, 5);
zeros.print();
SquareMatrix sqzeros = SquareMatrix::zeros(5);
sqzeros.print();
TESTCASE_END();

TESTCASE_BEGIN("Копирование и сложение");
SquareMatrix m = SquareMatrix::fill(5, 1);
SquareMatrix copy(m);
m.plus(copy.plus(SquareMatrix::fill(5, 1))).print();
TESTCASE_END();

TESTCASE_BEGIN("Префиксный инкремент");
SquareMatrix a = SquareMatrix::fill(5, 1);
(++a).print();
a.print();
TESTCASE_END();

TESTCASE_BEGIN("Постфиксный инкремент");
SquareMatrix b = SquareMatrix::fill(5, 1);
(b++).print();
b.print();
TESTCASE_END();

TESTCASE_BEGIN("Присваивание");
SquareMatrix c = SquareMatrix::fill(5, 1);
SquareMatrix d = SquareMatrix::identity(5);

```

```

SquareMatrix e;
e = c;
e.print();
e = d;
e.print();
TESTCASE_END();

TESTCASE_BEGIN("operator+");
SquareMatrix c = SquareMatrix::fill(5, 1);
SquareMatrix d = SquareMatrix::identity(5);
(c + d).print();
TESTCASE_END();

TESTCASE_BEGIN("operator-");
SquareMatrix c = SquareMatrix::fill(5, 1);
SquareMatrix d = SquareMatrix::identity(5);
(c - d).print();
TESTCASE_END();

TESTCASE_BEGIN("To int");
std::cout << int(SquareMatrix::identity(5)) << std::endl;
TESTCASE_END();

TESTCASE_BEGIN("Out to text file");
File otf("./assets/out.txt", std::ios_base::out);
otf << SquareMatrix::identity(5);
otf.close();
TESTCASE_END();

TESTCASE_BEGIN("Out to binary file");
File obf("./assets/out.b", std::ios_base::binary | std::ios_base::out);
obf << SquareMatrix::identity(5);
obf.close();
TESTCASE_END();

TESTCASE_BEGIN("In from binary file");
Matrix mmm;
File ibf("./assets/in.b", std::ios_base::binary | std::ios_base::in);
ibf >> mmm;
ibf.close();
mmm.print();
TESTCASE_END();

TESTCASE_BEGIN("Overload vectors print");
VectorH vec = Matrix::fill(1, 10, 5);
vec.print();
reinterpret_cast<Matrix *>(&vec)->print();
TESTCASE_END();

TESTCASE_BEGIN("VectorH to VectorV");
VectorH vech = Matrix::fill(1, 10, 5);
vech.print();
vech.transpose().print();

```

```

TESTCASE_END();

return 0;
}

```

Листинг 13 – matrix.hpp

```

#include <cstdint>
#include <fstream>
#include <ostream>

#include "file.hpp"

#define MATRIX_DATATYPE double

class Matrix {
private:
    std::size_t _rows;
    std::size_t _cols;
    MATRIX_DATATYPE *_data;

    void _init(const std::size_t, const std::size_t);
    void _assert_indexes(const Matrix &);

public:
    Matrix();
    Matrix(const std::size_t, const std::size_t);
    // Matrix(std::size_t, std::size_t, MATRIX_DATATYPE *);
    // Matrix(std::size_t, std::size_t, MATRIX_DATATYPE **);
    Matrix(const Matrix &);
    ~Matrix();

    Matrix plus(const Matrix &) const;
    virtual void fprint(std::ostream &) const;
    void print() const;
    Matrix transpose() const;
    std::size_t getRows() const;
    std::size_t getCols() const;

    static Matrix fill(std::size_t, std::size_t, MATRIX_DATATYPE);
    static Matrix zeros(std::size_t, std::size_t);

    Matrix operator+() const;
    Matrix operator+(const Matrix &) const;
    MATRIX_DATATYPE operator()(std::size_t, std::size_t) const;
    MATRIX_DATATYPE &operator()(std::size_t, std::size_t);
    Matrix &operator++();
    Matrix operator++(int);
    Matrix &operator--();

```

```

Matrix operator--(int);

operator int();

Matrix &operator=(const Matrix &other) noexcept;

friend const Matrix operator-(const Matrix &);
friend const Matrix operator-(const Matrix &, const Matrix &);
friend const Matrix operator*(MATRIX_DATATYPE, const Matrix &);
friend const Matrix operator*(const Matrix &, MATRIX_DATATYPE);
friend const Matrix operator*(const Matrix &, const Matrix &);

friend File &operator>>(File &, Matrix &);
friend File &operator<<(File &, const Matrix &);
};

class SquareMatrix : public Matrix {
public:
    SquareMatrix() : SquareMatrix(0){};
    SquareMatrix(std::size_t size) : Matrix(size, size){};
    SquareMatrix(const Matrix &);
    SquareMatrix(const SquareMatrix &matrix)
        : Matrix(dynamic_cast<const Matrix &>(matrix)){};
    ~SquareMatrix() = default;

    static SquareMatrix fill(std::size_t, MATRIX_DATATYPE);
    static SquareMatrix zeros(std::size_t);
    static SquareMatrix diagonal(std::size_t, MATRIX_DATATYPE);
    static SquareMatrix identity(std::size_t);

    SquareMatrix operator=(const SquareMatrix &) noexcept;

    SquareMatrix transpose() const;
};

class VectorH;
class VectorV;

class VectorH : public Matrix {
public:
    VectorH() : VectorH(0){};
    VectorH(std::size_t size) : Matrix(1, size){};
    VectorH(const Matrix &);
    VectorH(const VectorH &matrix)
        : Matrix(dynamic_cast<const Matrix &>(matrix)){};
    ~VectorH() = default;

    VectorV transpose() const;

    void fprint(std::ostream &) const override;
};

class VectorV : public Matrix {

```

```

public:
VectorV() : VectorV(0){};
VectorV(std::size_t size) : Matrix(size, 1){};
VectorV(const Matrix &);
VectorV(const VectorH &matrix)
    : Matrix(dynamic_cast<const Matrix &>(matrix)){};
~VectorV() = default;

VectorH transpose() const;

void fprintf(std::ostream &) const override;
};

```

Листинг 14 – matrix.cpp

```

#include "matrix.hpp"

#include <cassert>
#include <cmath>
#include <cstdint>
#include <cstdint>
#include <cstdint>
#include <iomanip>
#include <ios>
#include <iostream>

#define _ASSERT_INDEXES(SELF, OTHER) \
    assert((SELF).getRows() == (OTHER).getRows() \
        && (SELF).getCols() == (OTHER).getCols());

void Matrix::_init(const std::size_t rows, const std::size_t cols) {
    _rows = rows;
    _cols = cols;
    size_t size = _rows * _cols;
    if (size > 0)
        _data = new MATRIX_DATATYPE[size];
}

void Matrix::_assert_indexes(const Matrix &other) {
    assert(_rows == other.getRows() && _cols == other.getCols());
}

Matrix::Matrix() : Matrix(0, 0) {}

Matrix::Matrix(const std::size_t rows, const std::size_t cols) {
    _init(rows, cols);
}

Matrix::Matrix(const Matrix &matrix) {
    std::size_t rows = matrix._rows;

```

```

std::size_t cols = matrix._cols;
std::size_t size = rows * cols;

_init(rows, cols);

for (std::size_t row = 0; row < rows; ++row) {
    for (std::size_t col = 0; col < cols; ++col) {
        operator()(row, col) = matrix(row, col);
    }
}
}

Matrix::~Matrix() {
    if (_rows * _cols > 0)
        delete[] _data;
}

Matrix Matrix::plus(const Matrix &other) const {
    _ASSERT_INDEXES(*this, other);

    Matrix result(_rows, _cols);

    for (std::size_t row = 0; row < _rows; ++row) {
        for (std::size_t col = 0; col < _cols; ++col) {
            result(row, col) = operator()(row, col) + other(row, col);
        }
    }

    return result;
}

Matrix Matrix::transpose() const {
    Matrix result(_cols, _rows);

    for (std::size_t row = 0; row < _cols; ++row) {
        for (std::size_t col = 0; col < _rows; ++col) {
            result(row, col) = operator()(col, row);
        }
    }

    return result;
}

void Matrix::fprint(std::ostream &stream) const {
    stream << "[";
    for (std::size_t row = 0; row < _rows; ++row) {
        if (row != 0)
            stream << " ";
        stream << "[";
        for (std::size_t col = 0; col < _cols; ++col) {
            stream << operator()(row, col);
            if (col + 1 < _cols)
                stream << ", ";
        }
        stream << "];";
    }
}

```

```

        if (row + 1 < _rows)
            stream << "," << std::endl;
    }
    stream << "]" << std::endl;
}

void Matrix::print() const {
    fprintf(std::cout);
}

std::size_t Matrix::getRows() const {
    return _rows;
}

std::size_t Matrix::getCols() const {
    return _cols;
}

Matrix Matrix::fill(std::size_t rows, std::size_t cols, MATRIX_DATATYPE value) {
    Matrix result(rows, cols);

    for (std::size_t row = 0; row < rows; ++row) {
        for (std::size_t col = 0; col < cols; ++col) {
            result(row, col) = value;
        }
    }
    return result;
}

// Matrix Matrix::diagonal(size_t size, MATRIX_DATATYPE value) {
//     Matrix result = Matrix::zeros(size);
//     for (size_t i = 0; i < result.getSize(); ++i) {
//         result(i, i) = value;
//     }
//     return result;
// }

// Matrix Matrix::identity(size_t size) {
//     return Matrix::diagonal(size, 1);
// }

Matrix Matrix::zeros(std::size_t rows, std::size_t cols) {
    return Matrix::fill(rows, cols, 0.0);
}

Matrix Matrix::operator+() const {
    return Matrix(*this);
}

Matrix Matrix::operator+(const Matrix &other) const {
    _ASSERT_INDEXES(*this, other);
    Matrix result(*this);

    for (std::size_t row = 0; row < _rows; ++row) {

```



```

        for (std::size_t col = 0; col < _cols; ++col) {
            result(row, col) += other(row, col);
        }
    }

    return result;
}

Matrix &Matrix::operator++() {
    for (std::size_t row = 0; row < _rows; ++row) {
        for (std::size_t col = 0; col < _cols; ++col) {
            operator()(row, col) = operator()(row, col) + 1;
        }
    }
    return *this;
}

Matrix Matrix::operator++(int) {
    Matrix result(*this);
    operator++;
    return result;
}

Matrix &Matrix::operator--() {
    for (std::size_t row = 0; row < _rows; ++row) {
        for (std::size_t col = 0; col < _cols; ++col) {
            operator()(row, col) = operator()(row, col) - 1;
        }
    }
    return *this;
}

Matrix Matrix::operator--(int) {
    Matrix result(*this);
    operator--;
    return result;
}

Matrix::operator int() {
    return _rows * _cols;
}

Matrix &Matrix::operator=(const Matrix &other) noexcept {
    if (&other == this)
        return *this;

    if (_rows * _cols > 0)
        delete[] _data;

    _rows = other.getRows();
    _cols = other.getCols();
    _data = new MATRIX_DATATYPE[_rows * _cols];

    for (std::size_t row = 0; row < _rows; ++row) {

```

```

        for (std::size_t col = 0; col < _cols; ++col) {
            operator()(row, col) = other(row, col);
        }
    }

    return *this;
}

MATRIX_DATATYPE Matrix::operator()(size_t row, size_t col) const {
    assert(row < _rows && col < _cols);
    return _data[row * _cols + col];
}

MATRIX_DATATYPE &Matrix::operator()(size_t row, size_t col) {
    assert(row < _rows && col < _cols);
    return _data[row * _cols + col];
}

const Matrix operator-(const Matrix &matrix) {
    return -1.0 * matrix;
}

const Matrix operator-(const Matrix &matrix, const Matrix &other) {
    _ASSERT_INDEXES(matrix, other);
    const Matrix o = -other;
    return matrix + o;
}

const Matrix operator*(MATRIX_DATATYPE value, const Matrix &matrix) {
    Matrix result(matrix);
    for (std::size_t row = 0; row < matrix._rows; ++row) {
        for (std::size_t col = 0; col < matrix._cols; ++col) {
            result(row, col) = matrix(row, col) * value;
        }
    }
    return result;
}

const Matrix operator*(const Matrix &matrix, MATRIX_DATATYPE value) {
    return value * matrix;
}

const Matrix operator*(const Matrix &matrix, const Matrix &other) {
    // TODO:
    return matrix;
}

File &operator>>(File &file, Matrix &matrix) {
    if ((file.getMode() & std::ios_base::in) <= 0) {
        return file;
    }
    std::fstream &fs = file.getFStream();
    if ((file.getMode() & std::ios_base::binary) > 0) {
        std::size_t rows, cols;

```

```

MATRIX_DATATYPE el;

fs.read((char *)&rows, sizeof(size_t));
fs.read((char *)&cols, sizeof(size_t));
matrix = Matrix(rows, cols);

for (std::size_t row = 0; row < rows; ++row) {
    for (std::size_t col = 0; col < cols; ++col) {
        fs.read((char *)&el, sizeof(MATRIX_DATATYPE));
        matrix(row, col) = el;
    }
}
} else {
    // TODO
}
return file;
}

File &operator<<(File &file, const Matrix &matrix) {
    if ((file.getMode() & std::ios_base::out) <= 0) {
        return file;
    }

    std::fstream &fs = file.getFStream();

    if ((file.getMode() & std::ios_base::binary) > 0) {
        fs.write((char *)&matrix._rows, sizeof(size_t));
        fs.write((char *)&matrix._cols, sizeof(size_t));
        for (std::size_t row = 0; row < matrix._rows; ++row) {
            for (std::size_t col = 0; col < matrix._cols; ++col) {
                MATRIX_DATATYPE el = matrix(row, col);
                fs.write((char *)&el, sizeof(MATRIX_DATATYPE));
            }
        }
    } else {
        matrix.fprint(fs);
    }
    return file;
}

SquareMatrix::SquareMatrix(const Matrix &matrix)
    : Matrix(dynamic_cast<const Matrix &>(matrix)) {
    assert(matrix.getRows() == matrix.getCols());
}

SquareMatrix SquareMatrix::fill(std::size_t size, MATRIX_DATATYPE value) {
    return SquareMatrix(Matrix::fill(size, size, value));
}

SquareMatrix SquareMatrix::zeros(std::size_t size) {
    return SquareMatrix::fill(size, 0);
}

SquareMatrix SquareMatrix::diagonal(std::size_t size, MATRIX_DATATYPE value) {

```

```

    SquareMatrix result = SquareMatrix::zeros(size);
    for (std::size_t i = 0; i < size; ++i) {
        result(i, i) = value;
    }
    return result;
}

SquareMatrix SquareMatrix::identity(std::size_t size) {
    return SquareMatrix::diagonal(size, 1);
}

SquareMatrix SquareMatrix::transpose() const {
    return SquareMatrix(Matrix::transpose());
}

SquareMatrix SquareMatrix::operator=(const SquareMatrix &other) noexcept {
    Matrix::operator=(other);
    return *this;
}

VectorH::VectorH(const Matrix &matrix)
    : Matrix(dynamic_cast<const Matrix &>(matrix)) {
    assert(matrix.getRows() == 1);
}

void VectorH::fprint(std::ostream &stream) const {
    std::cout << "[";
    std::size_t size = getCols();
    for (std::size_t i = 0; i < size; ++i) {
        std::cout << operator()(0, i);
        if (i + 1 < size)
            std::cout << ", ";
    }
    std::cout << "]" << std::endl;
}

VectorV::VectorV(const Matrix &matrix)
    : Matrix(dynamic_cast<const Matrix &>(matrix)) {
    assert(matrix.getCols() == 1);
}

void VectorV::fprint(std::ostream &stream) const {
    std::cout << "[";
    std::size_t size = getRows();
    for (std::size_t i = 0; i < size; ++i) {
        if (i != 0)
            stream << " ";
        std::cout << operator()(i, 0);
        if (i + 1 < size)
            std::cout << "," << std::endl;
    }
    std::cout << "]" << std::endl;
}

VectorV VectorH::transpose() const {
    return VectorV(Matrix::transpose());
}

```

```

}
VectorH VectorV::transpose() const {
    return VectorH(Matrix::transpose());
}

```

Листинг 15 – file.hpp

```

#include <fstream>
#include <ios>

class File {
private:
    int _mode;
    std::fstream _fs;

public:
    File(const char *, std::ios_base::openmode);
    ~File();
    void close();
    int getMode();
    std::fstream &getFStream();
};

```

Листинг 16 – file.cpp

```

#include "file.hpp"
#include <ios>

File::File(const char *filename, std::ios_base::openmode mode) : _mode(mode) {
    _fs.open(filename, mode);
}

File::~File() {
    close();
}

void File::close() {
    _fs.close();
}

int File::getMode() {
    return static_cast<int>(_mode);
}

std::fstream &File::getFStream() {
    return _fs;
}

```

```
}
```

4.5. Выводы

В данной работе были рассмотрены принципы объектно-ориентированного программирования: наследование и полиморфизм, получены знания о виртуальных функциях, позднем связывании и таблице виртуальных функций (*vtable*).

5. ЛАБОРАТОРНАЯ РАБОТА №5. СОЗДАНИЕ ДИНАМИЧЕСКОГО СПИСКА ОБЪЕКТОВ, СВЯЗАННЫХ НАСЛЕДОВАНИЕМ

5.1. Цель работы

Изучить работу с динамическими списочными структурами данных.

5.2. Задание на лабораторную работу

Реализовать с помощью классов динамическую списочную структуру, содержащую объекты классов, связанных наследованием. В классах реализовать методы добавления, удаления, вставки по номеру, удаления по номеру, поиска и просмотра всей структуры.

Дополнить демонстрационную программу так, чтобы она демонстрировала полиморфическое поведение классов. Исследовать, как реализуется механизм полиморфизма.

Структура данных: циклическая очередь, реализованная на двунаправленном списке.

Способ хранения объектов: ссылки на объекты.

5.3. Решение

Циклическая очередь реализуется как класс *CircularQueueOfMatrix*, хранящий указатель на первый элемент (ноду) *CircularQueueOfMatrixNode*, которая в свою очередь хранит указатели на предыдущий и следующий элементы, а так же ссылку на матрицу.

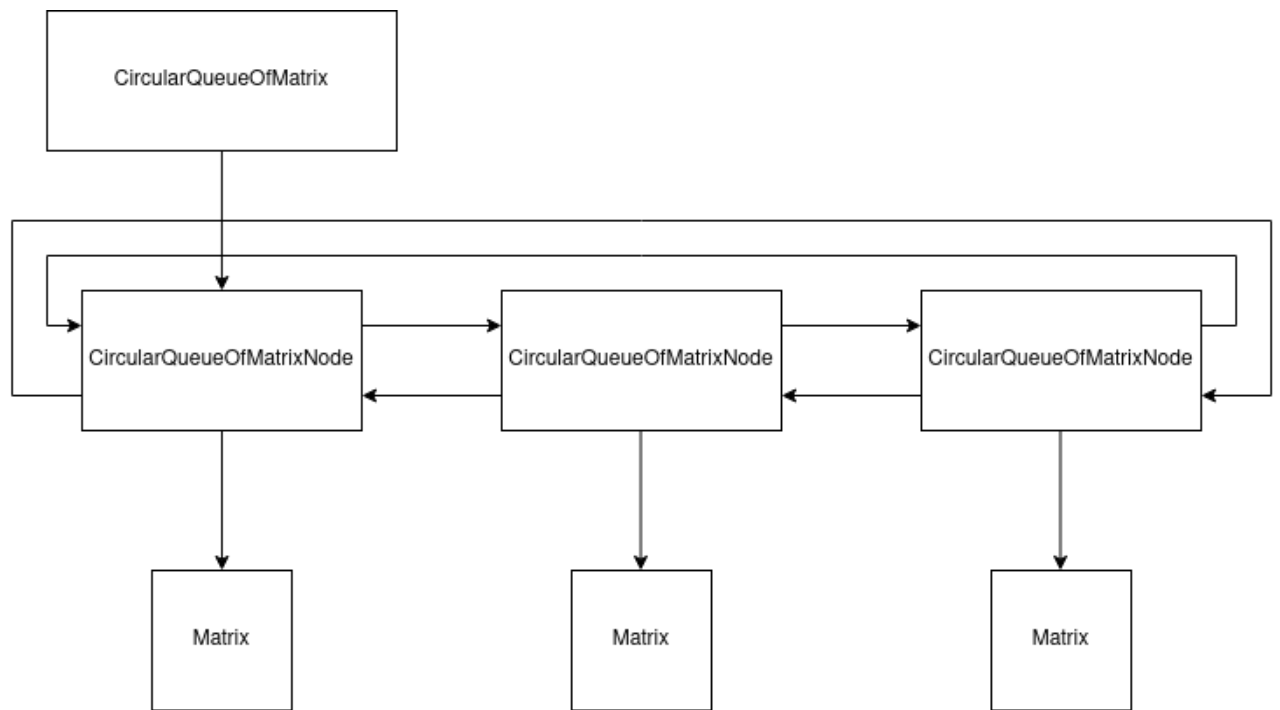


Рисунок 1 – Рисунок организации циклической очереди.

```

Test 19: CircularQueue
(0):
[[1, 0, 0, 0, 0],
 [0, 1, 0, 0, 0],
 [0, 0, 1, 0, 0],
 [0, 0, 0, 1, 0],
 [0, 0, 0, 0, 1]]
(1):
[0, 0, 0, 0, 0]
Queue is clear
% ~ /c/nstu-labs * term_3/lab_5 |
  
```

Рисунок 2 – Вывод работы программы.

5.4. Исходный код

Листинг 17 – main.cpp

```

#include "matrix.hpp"

#include <cstdint>
#include <cstdint>
#include <fstream>
#include <ios>
#include <iostream>
#include <ostream>

#include "circularqueue.hpp"
  
```



```

static int test_num = 0;

#define TESTCASE_BEGIN(TEXT)
    test_num++;
    std::cout << "Test " << test_num << ": " TEXT << std::endl;
    {
#define TESTCASE_END() }

int main() {
    TESTCASE_BEGIN("");
    SquareMatrix m(5);
    for (int i = 0; i < 5 * 5; ++i) {
        m(i / 5, i % 5) = i;
    }
    m.print();
    TESTCASE_END();

    TESTCASE_BEGIN("Транспонированная матрица");
    SquareMatrix m(5);
    for (int i = 0; i < 5 * 5; ++i) {
        m(i / 5, i % 5) = i;
    }
    m.print();
    SquareMatrix tr = m.transpose();
    tr.print();
    TESTCASE_END();

    TESTCASE_BEGIN("Заполнение матрицы");
    Matrix fill = Matrix::fill(5, 5, 5);
    fill.print();
    SquareMatrix sqfill = SquareMatrix::fill(5, 5);
    sqfill.print();
    TESTCASE_END();

    TESTCASE_BEGIN("Диагональная матрица");
    SquareMatrix diagonal = SquareMatrix::diagonal(5, 5);
    diagonal.print();
    TESTCASE_END();

    TESTCASE_BEGIN("Единичная матрица");
    SquareMatrix identity = SquareMatrix::identity(5);
    identity.print();
    TESTCASE_END();

    TESTCASE_BEGIN("Матрица нулей");
    SquareMatrix zeros = Matrix::zeros(5, 5);
    zeros.print();
    SquareMatrix sqzeros = SquareMatrix::zeros(5);
    sqzeros.print();
    TESTCASE_END();

    TESTCASE_BEGIN("Копирование и сложение");

```

```

SquareMatrix m = SquareMatrix::fill(5, 1);
SquareMatrix copy(m);
m.plus(copy.plus(SquareMatrix::fill(5, 1))).print();
TESTCASE_END();

TESTCASE_BEGIN("Префиксный инкремент");
SquareMatrix a = SquareMatrix::fill(5, 1);
(++a).print();
a.print();
TESTCASE_END();

TESTCASE_BEGIN("Постфиксный инкремент")
SquareMatrix b = SquareMatrix::fill(5, 1);
(b++).print();
b.print();
TESTCASE_END();

TESTCASE_BEGIN("Присваивание");
SquareMatrix c = SquareMatrix::fill(5, 1);
SquareMatrix d = SquareMatrix::identity(5);

SquareMatrix e;
e = c;
e.print();
e = d;
e.print();
TESTCASE_END();

TESTCASE_BEGIN("operator+");
SquareMatrix c = SquareMatrix::fill(5, 1);
SquareMatrix d = SquareMatrix::identity(5);
(c + d).print();
TESTCASE_END();

TESTCASE_BEGIN("operator-");
SquareMatrix c = SquareMatrix::fill(5, 1);
SquareMatrix d = SquareMatrix::identity(5);
(c - d).print();
TESTCASE_END();

TESTCASE_BEGIN("To int");
std::cout << int(SquareMatrix::identity(5)) << std::endl;
TESTCASE_END();

TESTCASE_BEGIN("Out to text file");
File otf("./assets/out.txt", std::ios_base::out);
otf << SquareMatrix::identity(5);
otf.close();
TESTCASE_END();

TESTCASE_BEGIN("Out to binary file");
File obf("./assets/out.b", std::ios_base::binary | std::ios_base::out);
obf << SquareMatrix::identity(5);
obf.close();

```

```

TESTCASE_END();

TESTCASE_BEGIN("In from binary file");
Matrix mmm;
File ibf("./assets/in.b", std::ios_base::binary | std::ios_base::in);
ibf >> mmm;
ibf.close();
mmm.print();
TESTCASE_END();

TESTCASE_BEGIN("Overload vectors print");
VectorH vec = Matrix::fill(1, 10, 5);
vec.print();
static_cast<Matrix &>(vec).print();
TESTCASE_END();

TESTCASE_BEGIN("VectorH to VectorV");
VectorH vech = Matrix::fill(1, 10, 5);
vech.print();
vech.transpose().print();
TESTCASE_END();

TESTCASE_BEGIN("CircularQueue");
CircularQueueOfMatrix cq;
Matrix **m = new Matrix *[2];
m[0] = new Matrix(5, 5);
*m[0] = SquareMatrix::identity(5);
m[1] = new VectorH(5);

for (std::size_t i = 0; i < 2; ++i) {
    cq.add(*m[i]);
}
cq.print();
for (std::size_t i = 0; i < 2; ++i) {
    cq.remove();
}
cq.print();
TESTCASE_END();

return 0;
}

```

Листинг 18 – matrix.hpp

```

#pragma once

#include <cstdint>
#include <cstdio>
#include <cstring>
#include <exception>
#include <fstream>
#include <limits>

```

```

#include <ostream>

#include "file.hpp"

#define MATRIX_DATATYPE double

class Matrix {
private:
    std::size_t _rows;
    std::size_t _cols;
    MATRIX_DATATYPE *_data;

    void _init(const std::size_t, const std::size_t);

public:
    Matrix();
    Matrix(const std::size_t, const std::size_t);
    // Matrix(std::size_t, std::size_t, MATRIX_DATATYPE *);
    // Matrix(std::size_t, std::size_t, MATRIX_DATATYPE **);
    Matrix(const Matrix &);
    ~Matrix();

    Matrix plus(const Matrix &) const;
    virtual void fprint(std::ostream &) const;
    void print() const;
    Matrix transpose() const;
    std::size_t getRows() const;
    std::size_t getCols() const;

    static Matrix fill(std::size_t, std::size_t, MATRIX_DATATYPE);
    static Matrix zeros(std::size_t, std::size_t);

    Matrix operator+() const;
    Matrix operator+(const Matrix &) const;
    MATRIX_DATATYPE operator()(std::size_t, std::size_t) const;
    MATRIX_DATATYPE &operator()(std::size_t, std::size_t);
    Matrix &operator++();
    Matrix operator++(int);
    Matrix &operator--();
    Matrix operator--(int);

    operator int();

    Matrix &operator=(const Matrix &other) noexcept;

    friend const Matrix operator-(const Matrix &);
    friend const Matrix operator-(const Matrix &, const Matrix &);
    friend const Matrix operator*(MATRIX_DATATYPE, const Matrix &);
    friend const Matrix operator*(const Matrix &, MATRIX_DATATYPE);
    friend const Matrix operator*(const Matrix &, const Matrix &);

```

```

    friend File &operator>>(File &, Matrix &);
    friend File &operator<<(File &, const Matrix &);
};

class SquareMatrix : public Matrix {
public:
    SquareMatrix() : SquareMatrix(0){};
    SquareMatrix(std::size_t size) : Matrix(size, size){};
    SquareMatrix(const Matrix &);
    SquareMatrix(const SquareMatrix &matrix)
        : Matrix(dynamic_cast<const Matrix &>(matrix)){};
    ~SquareMatrix() = default;

    static SquareMatrix fill(std::size_t, MATRIX_DATATYPE);
    static SquareMatrix zeros(std::size_t);
    static SquareMatrix diagonal(std::size_t, MATRIX_DATATYPE);
    static SquareMatrix identity(std::size_t);

    SquareMatrix operator=(const SquareMatrix &) noexcept;

    SquareMatrix transpose() const;
};

class VectorH;
class VectorV;

class VectorH : public Matrix {
public:
    VectorH() : VectorH(0){};
    VectorH(std::size_t size) : Matrix(1, size){};
    VectorH(const Matrix &);
    VectorH(const VectorH &matrix)
        : Matrix(dynamic_cast<const Matrix &>(matrix)){};
    ~VectorH() = default;

    VectorV transpose() const;

    void fprint(std::ostream &) const override;
};

class VectorV : public Matrix {
public:
    VectorV() : VectorV(0){};
    VectorV(std::size_t size) : Matrix(size, 1){};
    VectorV(const Matrix &);
    VectorV(const VectorH &matrix)
        : Matrix(dynamic_cast<const Matrix &>(matrix)){};
    ~VectorV() = default;

    VectorH transpose() const;

    void fprint(std::ostream &) const override;
};

```

Листинг 19 – matrix.cpp

```
#include "matrix.hpp"

#include <cassert>
#include <cmath>
#include <cstddef>
#include <cstdint>
#include <cstdlib>
#include <cstring>
#include <iomanip>
#include <ios>
#include <iostream>

#define _ASSERT_INDEXES(SELF, OTHER) \
    assert((SELF).getRows() == (OTHER).getRows() \
        && (SELF).getCols() == (OTHER).getCols());

void Matrix::_init(const std::size_t rows, const std::size_t cols) {
    _rows = rows;
    _cols = cols;
    size_t size = _rows * _cols;
    if (size > 0)
        _data = new MATRIX_DATATYPE[size];
}

Matrix::Matrix() : Matrix(0, 0) {}

Matrix::Matrix(const std::size_t rows, const std::size_t cols) {
    _init(rows, cols);
}

Matrix::Matrix(const Matrix &matrix) {
    std::size_t rows = matrix._rows;
    std::size_t cols = matrix._cols;
    std::size_t size = rows * cols;

    _init(rows, cols);

    for (std::size_t row = 0; row < rows; ++row) {
        for (std::size_t col = 0; col < cols; ++col) {
            operator()(row, col) = matrix(row, col);
        }
    }
}

Matrix::~Matrix() {
    if (_rows * _cols > 0)
        delete[] _data;
}

Matrix Matrix::plus(const Matrix &other) const {
```

```

    _ASSERT_INDEXES(*this, other);

    Matrix result(_rows, _cols);

    for (std::size_t row = 0; row < _rows; ++row) {
        for (std::size_t col = 0; col < _cols; ++col) {
            result(row, col) = operator()(row, col) + other(row, col);
        }
    }

    return result;
}

Matrix Matrix::transpose() const {
    Matrix result(_cols, _rows);

    for (std::size_t row = 0; row < _cols; ++row) {
        for (std::size_t col = 0; col < _rows; ++col) {
            result(row, col) = operator()(col, row);
        }
    }
    return result;
}

void Matrix::fprint(std::ostream &stream) const {
    stream << "[";
    for (std::size_t row = 0; row < _rows; ++row) {
        if (row != 0)
            stream << " ";
        stream << "[";
        for (std::size_t col = 0; col < _cols; ++col) {
            stream << operator()(row, col);
            if (col + 1 < _cols)
                stream << ", ";
        }
        stream << "]\n";
        if (row + 1 < _rows)
            stream << ",\n" << std::endl;
    }
    stream << "]" << std::endl;
}

void Matrix::print() const {
    fprint(std::cout);
}

std::size_t Matrix::getRows() const {
    return _rows;
}

std::size_t Matrix::getCols() const {
    return _cols;
}

```

```

Matrix Matrix::fill(std::size_t rows, std::size_t cols, MATRIX_DATATYPE value) {
    Matrix result(rows, cols);

    for (std::size_t row = 0; row < rows; ++row) {
        for (std::size_t col = 0; col < cols; ++col) {
            result(row, col) = value;
        }
    }
    return result;
}

// Matrix Matrix::diagonal(size_t size, MATRIX_DATATYPE value) {
//     Matrix result = Matrix::zeros(size);
//     for (size_t i = 0; i < result.getSize(); ++i) {
//         result(i, i) = value;
//     }
//     return result;
// }

// Matrix Matrix::identity(size_t size) {
//     return Matrix::diagonal(size, 1);
// }

Matrix Matrix::zeros(std::size_t rows, std::size_t cols) {
    return Matrix::fill(rows, cols, 0.0);
}

Matrix Matrix::operator+() const {
    return Matrix(*this);
}

Matrix Matrix::operator+(const Matrix &other) const {
    _ASSERT_INDEXES(*this, other);
    Matrix result(*this);

    for (std::size_t row = 0; row < _rows; ++row) {
        for (std::size_t col = 0; col < _cols; ++col) {
            result(row, col) += other(row, col);
        }
    }

    return result;
}

Matrix &Matrix::operator++() {
    for (std::size_t row = 0; row < _rows; ++row) {
        for (std::size_t col = 0; col < _cols; ++col) {
            operator()(row, col) = operator()(row, col) + 1;
        }
    }
    return *this;
}

Matrix Matrix::operator++(int) {

```



```

    Matrix result(*this);
    operator++();
    return result;
}

Matrix &Matrix::operator--() {
    for (std::size_t row = 0; row < _rows; ++row) {
        for (std::size_t col = 0; col < _cols; ++col) {
            operator()(row, col) = operator()(row, col) - 1;
        }
    }
    return *this;
}

Matrix Matrix::operator--(int) {
    Matrix result(*this);
    operator--();
    return result;
}

Matrix::operator int() {
    return _rows * _cols;
}

Matrix &Matrix::operator=(const Matrix &other) noexcept {
    if (&other == this)
        return *this;

    if (_rows * _cols > 0)
        delete[] _data;

    _rows = other.getRows();
    _cols = other.getCols();
    _data = new MATRIX_DATATYPE[_rows * _cols];

    for (std::size_t row = 0; row < _rows; ++row) {
        for (std::size_t col = 0; col < _cols; ++col) {
            operator()(row, col) = other(row, col);
        }
    }

    return *this;
}

MATRIX_DATATYPE Matrix::operator()(size_t row, size_t col) const {
    assert(row <= _rows && col <= _cols);
    return _data[row * _cols + col];
}

MATRIX_DATATYPE &Matrix::operator()(size_t row, size_t col) {
    assert(row <= _rows && col <= _cols);
    return _data[row * _cols + col];
}

```

```

const Matrix operator-(const Matrix &matrix) {
    return -1.0 * matrix;
}

const Matrix operator-(const Matrix &matrix, const Matrix &other) {
    _ASSERT_INDEXES(matrix, other);
    const Matrix o = -other;
    return matrix + o;
}

const Matrix operator*(MATRIX_DATATYPE value, const Matrix &matrix) {
    Matrix result(matrix);
    for (std::size_t row = 0; row < matrix._rows; ++row) {
        for (std::size_t col = 0; col < matrix._cols; ++col) {
            result(row, col) = matrix(row, col) * value;
        }
    }
    return result;
}

const Matrix operator*(const Matrix &matrix, MATRIX_DATATYPE value) {
    return value * matrix;
}

const Matrix operator*(const Matrix &matrix, const Matrix &other) {
    // TODO:
    return matrix;
}

File &operator>>(File &file, Matrix &matrix) {
    if ((file.getMode() & std::ios_base::in) <= 0) {
        return file;
    }
    std::fstream &fs = file.getFStream();
    if ((file.getMode() & std::ios_base::binary) > 0) {
        std::size_t rows, cols;
        MATRIX_DATATYPE el;

        fs.read((char *)&rows, sizeof(size_t));
        fs.read((char *)&cols, sizeof(size_t));
        matrix = Matrix(rows, cols);

        for (std::size_t row = 0; row < rows; ++row) {
            for (std::size_t col = 0; col < cols; ++col) {
                fs.read((char *)&el, sizeof(MATRIX_DATATYPE));
                matrix(row, col) = el;
            }
        }
    }
    else {
        // TODO
    }
    return file;
}

```

```

File &operator<<(File &file, const Matrix &matrix) {
    if ((file.getMode() & std::ios_base::out) <= 0) {
        return file;
    }

    std::fstream &fs = file.getFStream();

    if ((file.getMode() & std::ios_base::binary) > 0) {
        fs.write((char *)&matrix._rows, sizeof(size_t));
        fs.write((char *)&matrix._cols, sizeof(size_t));
        for (std::size_t row = 0; row < matrix._rows; ++row) {
            for (std::size_t col = 0; col < matrix._cols; ++col) {
                MATRIX_DATATYPE el = matrix(row, col);
                fs.write((char *)&el, sizeof(MATRIX_DATATYPE));
            }
        }
    } else {
        matrix.fprint(fs);
    }
    return file;
}

SquareMatrix::SquareMatrix(const Matrix &matrix)
    : Matrix(dynamic_cast<const Matrix &>(matrix)) {
    assert(matrix.getRows() == matrix.getCols());
}

SquareMatrix SquareMatrix::fill(std::size_t size, MATRIX_DATATYPE value) {
    return SquareMatrix(Matrix::fill(size, size, value));
}

SquareMatrix SquareMatrix::zeros(std::size_t size) {
    return SquareMatrix::fill(size, 0);
}

SquareMatrix SquareMatrix::diagonal(std::size_t size, MATRIX_DATATYPE value) {
    SquareMatrix result = SquareMatrix::zeros(size);
    for (std::size_t i = 0; i < size; ++i) {
        result(i, i) = value;
    }
    return result;
}

SquareMatrix SquareMatrix::identity(std::size_t size) {
    return SquareMatrix::diagonal(size, 1);
}

SquareMatrix SquareMatrix::transpose() const {
    return SquareMatrix(Matrix::transpose());
}

SquareMatrix SquareMatrix::operator=(const SquareMatrix &other) noexcept {
    Matrix::operator=(other);
    return *this;
}

```

```

VectorH::VectorH(const Matrix &matrix)
    : Matrix(dynamic_cast<const Matrix &>(matrix)) {
    assert(matrix.getRows() == 1);
}

void VectorH::fprint(std::ostream &stream) const {
    std::cout << "[";
    std::size_t size = getCols();
    for (std::size_t i = 0; i < size; ++i) {
        std::cout << operator()(0, i);
        if (i + 1 < size)
            std::cout << ", ";
    }
    std::cout << "]" << std::endl;
}

VectorV::VectorV(const Matrix &matrix)
    : Matrix(dynamic_cast<const Matrix &>(matrix)) {
    assert(matrix.getCols() == 1);
}

void VectorV::fprint(std::ostream &stream) const {
    std::cout << "[";
    std::size_t size = getRows();
    for (std::size_t i = 0; i < size; ++i) {
        if (i != 0)
            stream << " ";
        std::cout << operator()(i, 0);
        if (i + 1 < size)
            std::cout << "," << std::endl;
    }
    std::cout << "]" << std::endl;
}

VectorV VectorH::transpose() const {
    return VectorV(Matrix::transpose());
}

VectorH VectorV::transpose() const {
    return VectorH(Matrix::transpose());
}

```

Листинг 20 – file.hpp

```

#include <fstream>
#include <ios>

class File {
private:
    int _mode;
    std::fstream _fs;

```

```

    public:
    File(const char *, std::ios_base::openmode);
    ~File();
    void close();
    int getMode();
    std::fstream &getFStream();
};

```

Листинг 21 – file.cpp

```

#include "file.hpp"

#include <ios>

File::File(const char *filename, std::ios_base::openmode mode) : _mode(mode) {
    _fs.open(filename, mode);
}

File::~File() {
    close();
}

void File::close() {
    _fs.close();
}

int File::getMode() {
    return static_cast<int>(_mode);
}

std::fstream &File::getFStream() {
    return _fs;
}

```

Листинг 22 – circularqueue.hpp

```

#pragma once

#include "matrix.hpp"
#include <ostream>

class CircularQueueOfMatrixNode {
    private:
    CircularQueueOfMatrixNode *_prev;
    CircularQueueOfMatrixNode *_next;
    Matrix &_data;
};

```

```

public:
CircularQueueOfMatrixNode(Matrix &);
~CircularQueueOfMatrixNode() = default;

CircularQueueOfMatrixNode *getPrev();
void setPrev(CircularQueueOfMatrixNode *);
CircularQueueOfMatrixNode *getNext();
void setNext(CircularQueueOfMatrixNode *);
Matrix &getMatrix();
};

class CircularQueueOfMatrix {
private:
CircularQueueOfMatrixNode *_head;

public:
CircularQueueOfMatrix();
~CircularQueueOfMatrix();
void add(Matrix &);
Matrix &remove();
void fprint(std::ostream &);
void print();
};

```

Листинг 23 – circularqueue.cpp

```

#include "circularqueue.hpp"
#include <cassert>
#include <iostream>
#include <ostream>

CircularQueueOfMatrixNode::CircularQueueOfMatrixNode(Matrix &_matrix)
    : _data(_matrix), _next(nullptr), _prev(nullptr) {}

CircularQueueOfMatrixNode *CircularQueueOfMatrixNode::getPrev() {
    return _prev;
}

void CircularQueueOfMatrixNode::setPrev(CircularQueueOfMatrixNode *prev) {
    _prev = prev;
}

CircularQueueOfMatrixNode *CircularQueueOfMatrixNode::getNext() {
    return _next;
}

void CircularQueueOfMatrixNode::setNext(CircularQueueOfMatrixNode *next) {
    _next = next;
}

Matrix &CircularQueueOfMatrixNode::getMatrix() {

```

```

    return _data;
}

CircularQueueOfMatrix::CircularQueueOfMatrix() : _head(nullptr) {}

CircularQueueOfMatrix::~CircularQueueOfMatrix() {
    CircularQueueOfMatrixNode *iterator = _head, *next;
    do {
        if (!iterator)
            break;
        next = iterator->getNext();
        delete iterator;
        iterator = next;
    } while (iterator != _head);
}

void CircularQueueOfMatrix::add(Matrix &m) {
    CircularQueueOfMatrixNode *node = new CircularQueueOfMatrixNode(m);
    if (_head) {
        node->setNext(_head);
        node->setPrev(_head->getPrev());
        _head->getPrev()->setNext(node);
        _head->setPrev(node);
    } else {
        node->setNext(node);
        node->setPrev(node);
        _head = node;
    }
}

Matrix &CircularQueueOfMatrix::remove() {
    assert(_head != nullptr); // can be replaced with exceptions
    Matrix &result = _head->getMatrix();
    CircularQueueOfMatrixNode *next = _head->getNext(), *prev = _head->getPrev();
    if (_head == next) {
        delete _head;
        _head = nullptr;
        return result;
    }
    prev->setNext(next);
    next->setPrev(prev);

    delete _head;
    _head = next;
    return result;
}

void CircularQueueOfMatrix::fprint(std::ostream &os) {
    int num = 0;
    CircularQueueOfMatrixNode *iterator = _head;
    do {
        if (!iterator) {
            os << "Queue is clear" << std::endl;
            break;

```

```

    }
    os << "(" << num++ << "): " << std::endl;
    iterator->getMatrix().fprint(os);
    iterator = iterator->getNext();
} while (iterator != _head);
}

void CircularQueueOfMatrix::print() {
    fprint(std::cout);
}

```

5.5. Выводы

В этой лабораторной работе мы познакомились с разработкой полиморфных структур данных, при создании которых необходимо четко выделять интерфейсы для правильного обобщения хранимых в них типов.

6. ЛАБОРАТОРНАЯ РАБОТА №6. ОБРАБОТКА ИСКЛЮЧИТЕЛЬНЫХ СИТУАЦИЙ

6.1. Цель работы

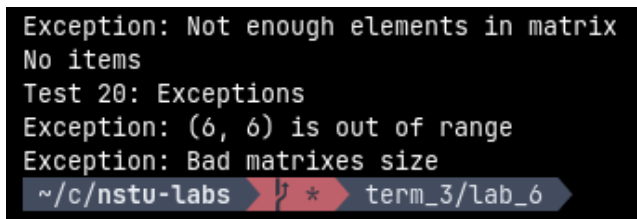
Изучить механизм обработки исключительных ситуаций в объектно-ориентированных программах.

6.2. Задание на лабораторную работу

Добавить в классы и демонстрационную программу обработку исключений при возникновении ошибок: недостатка памяти, выхода за пределы диапазона допустимых значений и т.д. Дополнить демонстрационную программу так, чтобы она демонстрировала обработку исключений.

6.3. Решение

Исключение – это событие, происходящее в момент выполнения программы, приводящее к остановке выполнения программы в случае, если оно не будет обработано. Обработка исключений обеспечивает способ передачи управления и информации из некоторой точки выполнения программы в обработчик, связанный с ранее пройденной точкой (другими словами, обработка исключений передает управление вверх по стеку вызовов). Исключения, реализованные в данной лабораторной работе являются наследниками класса `std::exception`, их обработка была добавлена в демонстрационную программу, а их определение, реализация и вызовы в исходные коды классов (Матрицы и очередь).



```
Exception: Not enough elements in matrix
No items
Test 20: Exceptions
Exception: (6, 6) is out of range
Exception: Bad matrixes size
~/c/nstu-labs * term_3/lab_6
```

Рисунок 1 – Вывод работы программы.

6.4. Исходный код

Листинг 24 – main.cpp

```
#include "matrix.hpp"

#include <cstdint>
#include <cstdlib>
```

```

#include <fstream>
#include <ios>
#include <iostream>
#include <ostream>

#include "circularqueue.hpp"

static int test_num = 0;

#define TESTCASE_BEGIN(TEXT)
    test_num++;
    std::cout << "Test " << test_num << ": " TEXT << std::endl;
    {
#define TESTCASE_END() }

int main() {
    TESTCASE_BEGIN("");
    SquareMatrix m(5);
    for (int i = 0; i < 5 * 5; ++i) {
        m(i / 5, i % 5) = i;
    }
    m.print();
    TESTCASE_END();

    TESTCASE_BEGIN("Транспонированная матрица");
    SquareMatrix m(5);
    for (int i = 0; i < 5 * 5; ++i) {
        m(i / 5, i % 5) = i;
    }
    m.print();
    SquareMatrix tr = m.transpose();
    tr.print();
    TESTCASE_END();

    TESTCASE_BEGIN("Заполнение матрицы");
    Matrix fill = Matrix::fill(5, 5, 5);
    fill.print();
    SquareMatrix sqfill = SquareMatrix::fill(5, 5);
    sqfill.print();
    TESTCASE_END();

    TESTCASE_BEGIN("Диагональная матрица");
    SquareMatrix diagonal = SquareMatrix::diagonal(5, 5);
    diagonal.print();
    TESTCASE_END();

    TESTCASE_BEGIN("Единичная матрица");
    SquareMatrix identity = SquareMatrix::identity(5);
    identity.print();
    TESTCASE_END();

    TESTCASE_BEGIN("Матрица нулей");

```

```

SquareMatrix zeros = Matrix::zeros(5, 5);
zeros.print();
SquareMatrix sqzeros = SquareMatrix::zeros(5);
sqzeros.print();
TESTCASE_END();

TESTCASE_BEGIN("Копирование и сложение");
SquareMatrix m = SquareMatrix::fill(5, 1);
SquareMatrix copy(m);
m.plus(copy.plus(SquareMatrix::fill(5, 1))).print();
TESTCASE_END();

TESTCASE_BEGIN("Префиксный инкремент");
SquareMatrix a = SquareMatrix::fill(5, 1);
(++a).print();
a.print();
TESTCASE_END();

TESTCASE_BEGIN("Постфиксный инкремент")
SquareMatrix b = SquareMatrix::fill(5, 1);
(b++).print();
b.print();
TESTCASE_END();

TESTCASE_BEGIN("Присваивание");
SquareMatrix c = SquareMatrix::fill(5, 1);
SquareMatrix d = SquareMatrix::identity(5);

SquareMatrix e;
e = c;
e.print();
e = d;
e.print();
TESTCASE_END();

TESTCASE_BEGIN("operator+");
SquareMatrix c = SquareMatrix::fill(5, 1);
SquareMatrix d = SquareMatrix::identity(5);
(c + d).print();
TESTCASE_END();

TESTCASE_BEGIN("operator-");
SquareMatrix c = SquareMatrix::fill(5, 1);
SquareMatrix d = SquareMatrix::identity(5);
(c - d).print();
TESTCASE_END();

TESTCASE_BEGIN("To int");
std::cout << int(SquareMatrix::identity(5)) << std::endl;
TESTCASE_END();

TESTCASE_BEGIN("Out to text file");
File otf("./assets/out.txt", std::ios_base::out);
otf << SquareMatrix::identity(5);

```

```

otf.close();
TESTCASE_END();

TESTCASE_BEGIN("Out to binary file");
File obf("./assets/out.b", std::ios_base::binary | std::ios_base::out);
obf << SquareMatrix::identity(5);
obf.close();
TESTCASE_END();

TESTCASE_BEGIN("In from binary file");
Matrix mmm;
File ibf("./assets/in.b", std::ios_base::binary | std::ios_base::in);
ibf >> mmm;
ibf.close();
mmm.print();
TESTCASE_END();

TESTCASE_BEGIN("Overload vectors print");
VectorH vec = Matrix::fill(1, 10, 5);
vec.print();
static_cast<Matrix &>(vec).print();
TESTCASE_END();

TESTCASE_BEGIN("VectorH to VectorV");
VectorH vech = Matrix::fill(1, 10, 5);
vech.print();
vech.transpose().print();
TESTCASE_END();

TESTCASE_BEGIN("CircularDoubleLinkedList");
CircularQueueOfMatrix cq;
Matrix *m1 = new Matrix(5, 5);
*m1 = Matrix::fill(5, 5, 5);
Matrix *m2 = new VectorH(5);

cq.add(*m1);
cq.add(*m2);

cq.print();
cq.remove();
cq.print();
cq.remove();
cq.print();
try {
    cq.remove();
} catch (CircularQueueOfMatrix::NotEnoughOfItems &e) {
    std::cout << "Exception: " << e.what() << std::endl;
}

cq.print();
TESTCASE_END();

TESTCASE_BEGIN("Exceptions")
Matrix m = Matrix::fill(5, 5, 5);

```

```

    try {
        m(6, 6);
    } catch (Matrix::OutOfRange &e) {
        std::cout << "Exception: " << e.what() << std::endl;
    }
    try {
        VectorH vech(m);
    } catch (Matrix::BadSize &e) {
        std::cout << "Exception: " << e.what() << std::endl;
    }
    TESTCASE_END();

    return 0;
}

```

Листинг 25 – matrix.hpp

```

#pragma once

#include <cstddef>
#include <cstdio>
#include <cstring>
#include <exception>
#include <fstream>
#include <limits>
#include <ostream>

#include "file.hpp"

#define MATRIX_DATATYPE double

class Matrix {
private:
    std::size_t _rows;
    std::size_t _cols;
    MATRIX_DATATYPE *_data;

    void _init(const std::size_t, const std::size_t);

public:
    Matrix();
    Matrix(const std::size_t, const std::size_t);
    // Matrix(std::size_t, std::size_t, MATRIX_DATATYPE *);
    // Matrix(std::size_t, std::size_t, MATRIX_DATATYPE **);
    Matrix(const Matrix &);
    ~Matrix();

    Matrix plus(const Matrix &) const;
    virtual void fprintf(std::ostream &) const;

```

```

void print() const;
Matrix transpose() const;
std::size_t getRows() const;
std::size_t getCols() const;

static Matrix fill(std::size_t, std::size_t, MATRIX_DATATYPE);
static Matrix zeros(std::size_t, std::size_t);

Matrix operator+() const;
Matrix operator+(const Matrix &) const;
MATRIX_DATATYPE operator()(std::size_t, std::size_t) const;
MATRIX_DATATYPE &operator()(std::size_t, std::size_t);
Matrix &operator++();
Matrix operator++(int);
Matrix &operator--();
Matrix operator--(int);

operator int();

Matrix &operator=(const Matrix &other) noexcept;

friend const Matrix operator-(const Matrix &);
friend const Matrix operator-(const Matrix &, const Matrix &);
friend const Matrix operator*(MATRIX_DATATYPE, const Matrix &);
friend const Matrix operator*(const Matrix &, MATRIX_DATATYPE);
friend const Matrix operator*(const Matrix &, const Matrix &);

friend File &operator>>(File &, Matrix &);
friend File &operator<<(File &, const Matrix &);

class OutOfRange : public std::exception {
private:
    std::size_t _row, _col;
    char *_message;

public:
    OutOfRange();
    OutOfRange(std::size_t row, std::size_t col);
    ~OutOfRange();
    const char *what() const noexcept;
};

class BadSize : public std::exception {
public:
    const char *what() const noexcept;
};

class SquareMatrix : public Matrix {
public:
    SquareMatrix() : SquareMatrix(0){};
    SquareMatrix(std::size_t size) : Matrix(size, size){};
    SquareMatrix(const Matrix &);
    SquareMatrix(const SquareMatrix &matrix)

```

```

        : Matrix(dynamic_cast<const Matrix &>(matrix)){};
~SquareMatrix() = default;

static SquareMatrix fill(std::size_t, MATRIX_DATATYPE);
static SquareMatrix zeros(std::size_t);
static SquareMatrix diagonal(std::size_t, MATRIX_DATATYPE);
static SquareMatrix identity(std::size_t);

SquareMatrix operator=(const SquareMatrix &) noexcept;

SquareMatrix transpose() const;
};

class VectorH;
class VectorV;

class VectorH : public Matrix {
public:
    VectorH() : VectorH(0){};
    VectorH(std::size_t size) : Matrix(1, size){};
    VectorH(const Matrix &);
    VectorH(const VectorH &matrix)
        : Matrix(dynamic_cast<const Matrix &>(matrix)){};
    ~VectorH() = default;

    VectorV transpose() const;

    void fprint(std::ostream &) const override;
};

class VectorV : public Matrix {
public:
    VectorV() : VectorV(0){};
    VectorV(std::size_t size) : Matrix(size, 1){};
    VectorV(const Matrix &);
    VectorV(const VectorH &matrix)
        : Matrix(dynamic_cast<const Matrix &>(matrix)){};
    ~VectorV() = default;

    VectorH transpose() const;

    void fprint(std::ostream &) const override;
};

```

Листинг 26 – matrix.cpp

```

#include "matrix.hpp"

#include <cassert>
#include <cmath>
#include <cstdint>

```

```

#include <cstdint>
#include <cstdlib>
#include <cstring>
#include <iomanip>
#include <ios>
#include <iostream>

Matrix::Matrix() : Matrix(0, 0) {}

Matrix::Matrix(const std::size_t rows, const std::size_t cols) {
    _init(rows, cols);
}

Matrix::Matrix(const Matrix &matrix) {
    std::size_t rows = matrix._rows;
    std::size_t cols = matrix._cols;
    std::size_t size = rows * cols;

    _init(rows, cols);

    for (std::size_t row = 0; row < rows; ++row) {
        for (std::size_t col = 0; col < cols; ++col) {
            operator()(row, col) = matrix(row, col);
        }
    }
}

Matrix::~Matrix() {
    if (_rows * _cols > 0)
        delete[] _data;
}

Matrix Matrix::plus(const Matrix &other) const {
    if (getRows() != other.getRows() || getCols() != other.getCols())
        throw BadSize();

    Matrix result(_rows, _cols);

    for (std::size_t row = 0; row < _rows; ++row) {
        for (std::size_t col = 0; col < _cols; ++col) {
            result(row, col) = operator()(row, col) + other(row, col);
        }
    }

    return result;
}

Matrix Matrix::transpose() const {
    Matrix result(_cols, _rows);

    for (std::size_t row = 0; row < _cols; ++row) {
        for (std::size_t col = 0; col < _rows; ++col) {
            result(row, col) = operator()(col, row);
        }
    }
}

```



```

    }
}
return result;
}

void Matrix::fprint(std::ostream &stream) const {
    stream << "[";
    for (std::size_t row = 0; row < _rows; ++row) {
        if (row != 0)
            stream << " ";
        stream << "[";
        for (std::size_t col = 0; col < _cols; ++col) {
            stream << operator()(row, col);
            if (col + 1 < _cols)
                stream << ", ";
        }
        stream << "];"
        if (row + 1 < _rows)
            stream << ", " << std::endl;
    }
    stream << "]" << std::endl;
}

void Matrix::print() const {
    fprint(std::cout);
}

std::size_t Matrix::getRows() const {
    return _rows;
}

std::size_t Matrix::getCols() const {
    return _cols;
}

Matrix Matrix::fill(std::size_t rows, std::size_t cols, MATRIX_DATATYPE value) {
    Matrix result(rows, cols);

    for (std::size_t row = 0; row < rows; ++row) {
        for (std::size_t col = 0; col < cols; ++col) {
            result(row, col) = value;
        }
    }
    return result;
}

Matrix Matrix::zeros(std::size_t rows, std::size_t cols) {
    return Matrix::fill(rows, cols, 0.0);
}

Matrix Matrix::operator+() const {
    return Matrix(*this);
}

```

```

Matrix Matrix::operator+(const Matrix &other) const {
    if (getRows() != other.getRows() || getCols() != other.getCols())
        throw BadSize();
    Matrix result(*this);

    for (std::size_t row = 0; row < _rows; ++row) {
        for (std::size_t col = 0; col < _cols; ++col) {
            result(row, col) += other(row, col);
        }
    }

    return result;
}

Matrix &Matrix::operator++() {
    for (std::size_t row = 0; row < _rows; ++row) {
        for (std::size_t col = 0; col < _cols; ++col) {
            operator()(row, col) = operator()(row, col) + 1;
        }
    }
    return *this;
}

Matrix Matrix::operator++(int) {
    Matrix result(*this);
    operator++();
    return result;
}

Matrix &Matrix::operator--() {
    for (std::size_t row = 0; row < _rows; ++row) {
        for (std::size_t col = 0; col < _cols; ++col) {
            operator()(row, col) = operator()(row, col) - 1;
        }
    }
    return *this;
}

Matrix Matrix::operator--(int) {
    Matrix result(*this);
    operator--();
    return result;
}

Matrix::operator int() {
    return _rows * _cols;
}

Matrix &Matrix::operator=(const Matrix &other) noexcept {
    if (&other == this)
        return *this;

    if (_rows * _cols > 0)
        delete[] _data;

```

```

    _rows = other.getRows();
    _cols = other.getCols();
    _data = new MATRIX_DATATYPE[_rows * _cols];

    for (std::size_t row = 0; row < _rows; ++row) {
        for (std::size_t col = 0; col < _cols; ++col) {
            operator()(row, col) = other(row, col);
        }
    }

    return *this;
}

MATRIX_DATATYPE Matrix::operator()(size_t row, size_t col) const {
    if (row >= _rows || col >= _cols)
        throw OutOfRange(row, col);

    return _data[row * _cols + col];
}

MATRIX_DATATYPE &Matrix::operator()(size_t row, size_t col) {
    if (row >= _rows || col >= _cols)
        throw OutOfRange(row, col);
    return _data[row * _cols + col];
}

const Matrix operator-(const Matrix &matrix) {
    return -1.0 * matrix;
}

const Matrix operator-(const Matrix &matrix, const Matrix &other) {
    if (matrix.getRows() != other.getRows()
        || matrix.getCols() != other.getCols())
        throw Matrix::BadSize();
    const Matrix o = -other;
    return matrix + o;
}

const Matrix operator*(MATRIX_DATATYPE value, const Matrix &matrix) {
    Matrix result(matrix);
    for (std::size_t row = 0; row < matrix._rows; ++row) {
        for (std::size_t col = 0; col < matrix._cols; ++col) {
            result(row, col) = matrix(row, col) * value;
        }
    }
    return result;
}

const Matrix operator*(const Matrix &matrix, MATRIX_DATATYPE value) {
    return value * matrix;
}

const Matrix operator*(const Matrix &matrix, const Matrix &other) {

```

```

    // TODO:
    return matrix;
}

File &operator>>(File &file, Matrix &matrix) {
    if ((file.getMode() & std::ios_base::in) <= 0) {
        return file;
    }
    std::fstream &fs = file.getFStream();
    if ((file.getMode() & std::ios_base::binary) > 0) {
        std::size_t rows, cols;
        MATRIX_DATATYPE el;

        fs.read((char *)&rows, sizeof(size_t));
        fs.read((char *)&cols, sizeof(size_t));
        matrix = Matrix(rows, cols);

        for (std::size_t row = 0; row < rows; ++row) {
            for (std::size_t col = 0; col < cols; ++col) {
                fs.read((char *)&el, sizeof(MATRIX_DATATYPE));
                matrix(row, col) = el;
            }
        }
    } else {
        // TODO
    }
    return file;
}

File &operator<<(File &file, const Matrix &matrix) {
    if ((file.getMode() & std::ios_base::out) <= 0) {
        return file;
    }

    std::fstream &fs = file.getFStream();

    if ((file.getMode() & std::ios_base::binary) > 0) {
        fs.write((char *)&matrix._rows, sizeof(size_t));
        fs.write((char *)&matrix._cols, sizeof(size_t));
        for (std::size_t row = 0; row < matrix._rows; ++row) {
            for (std::size_t col = 0; col < matrix._cols; ++col) {
                MATRIX_DATATYPE el = matrix(row, col);
                fs.write((char *)&el, sizeof(MATRIX_DATATYPE));
            }
        }
    } else {
        matrix.fprint(fs);
    }
    return file;
}

SquareMatrix::SquareMatrix(const Matrix &matrix)
    : Matrix(dynamic_cast<const Matrix &>(matrix)) {
    if (matrix.getRows() != matrix.getCols())

```

```

        throw BadSize();
    }

    SquareMatrix SquareMatrix::fill(std::size_t size, MATRIX_DATATYPE value) {
        return SquareMatrix(Matrix::fill(size, size, value));
    }

    SquareMatrix SquareMatrix::zeros(std::size_t size) {
        return SquareMatrix::fill(size, 0);
    }

    SquareMatrix SquareMatrix::diagonal(std::size_t size, MATRIX_DATATYPE value) {
        SquareMatrix result = SquareMatrix::zeros(size);
        for (std::size_t i = 0; i < size; ++i) {
            result(i, i) = value;
        }
        return result;
    }

    SquareMatrix SquareMatrix::identity(std::size_t size) {
        return SquareMatrix::diagonal(size, 1);
    }

    SquareMatrix SquareMatrix::transpose() const {
        return SquareMatrix(Matrix::transpose());
    }

    SquareMatrix SquareMatrix::operator=(const SquareMatrix &other) noexcept {
        Matrix::operator=(other);
        return *this;
    }

    VectorH::VectorH(const Matrix &matrix)
        : Matrix(dynamic_cast<const Matrix &>(matrix)) {
        if (matrix.getRows() != 1)
            throw BadSize();
    }

    void VectorH::fprint(std::ostream &stream) const {
        std::cout << "[";
        std::size_t size = getCols();
        for (std::size_t i = 0; i < size; ++i) {
            std::cout << operator()(0, i);
            if (i + 1 < size)
                std::cout << ", ";
        }
        std::cout << "]" << std::endl;
    }

    VectorV::VectorV(const Matrix &matrix)
        : Matrix(dynamic_cast<const Matrix &>(matrix)) {
        if (matrix.getCols() != 1)
            throw BadSize();
    }

    void VectorV::fprint(std::ostream &stream) const {

```

```

std::cout << "[";
std::size_t size = getRows();
for (std::size_t i = 0; i < size; ++i) {
    if (i != 0)
        stream << " ";
    std::cout << operator()(i, 0);
    if (i + 1 < size)
        std::cout << "," << std::endl;
}
std::cout << "]" << std::endl;
}

VectorV VectorH::transpose() const {
    return VectorV(Matrix::transpose());
}
VectorH VectorV::transpose() const {
    return VectorH(Matrix::transpose());
}

Matrix::OutOfRange::OutOfRange() {
    _message = new char[12];
    strcpy(_message, "out of range");
};

Matrix::OutOfRange::OutOfRange(std::size_t row, std::size_t col)
    : _row(row), _col(col) {
    _message = new char[20 + 2 * std::numeric_limits<std::size_t>().max_digits10];
    strcpy(_message, "(");
    sprintf(_message + 1, "%lu", _row);
    strcat(_message + strlen(_message), ", ");
    sprintf(_message + strlen(_message), "%lu", _col);
    strcat(_message + strlen(_message), ") is out of range");
}

Matrix::OutOfRange::~~OutOfRange() {
    delete _message;
}

const char *Matrix::OutOfRange::what() const noexcept {
    return _message;
}

void Matrix::_init(const std::size_t rows, const std::size_t cols) {
    _rows = rows;
    _cols = cols;
    size_t size = _rows * _cols;
    if (size > 0)
        _data = new MATRIX_DATATYPE[size];
}

const char *Matrix::BadSize::what() const noexcept {
    return "Bad matrixes size";
}

```

Листинг 27 – file.hpp

```
#include <fstream>
#include <ios>

class File {
private:
    int _mode;
    std::fstream _fs;

public:
    File(const char *, std::ios_base::openmode);
    ~File();
    void close();
    int getMode();
    std::fstream &getFStream();
};
```

Листинг 28 – file.cpp

```
#include "file.hpp"

#include <ios>

File::File(const char *filename, std::ios_base::openmode mode) : _mode(mode) {
    _fs.open(filename, mode);
}

File::~File() {
    close();
}

void File::close() {
    _fs.close();
}

int File::getMode() {
    return static_cast<int>(_mode);
}

std::fstream &File::getFStream() {
    return _fs;
}
```

Листинг 29 – circularqueue.hpp

```

#pragma once

#include "matrix.hpp"
#include <exception>
#include <ostream>

class CircularQueueOfMatrixNode {
private:
    CircularQueueOfMatrixNode *_prev;
    CircularQueueOfMatrixNode *_next;
    Matrix &_data;

public:
    CircularQueueOfMatrixNode(Matrix &);
    ~CircularQueueOfMatrixNode() = default;

    CircularQueueOfMatrixNode *_getPrev();
    void setPrev(CircularQueueOfMatrixNode *);
    CircularQueueOfMatrixNode *_getNext();
    void setNext(CircularQueueOfMatrixNode *);
    Matrix &getMatrix();
};

class CircularQueueOfMatrix {
private:
    CircularQueueOfMatrixNode *_head;

public:
    CircularQueueOfMatrix();
    ~CircularQueueOfMatrix();
    void add(Matrix &);
    Matrix &remove();
    void fprintf(std::ostream &);
    void print();

    class NotEnoughOfItems : public std::exception {
public:
        const char *what() const noexcept;
    };
};

```

Листинг 30 – circularqueue.cpp

```

#include "circularqueue.hpp"
#include <iostream>

CircularQueueOfMatrixNode::CircularQueueOfMatrixNode(Matrix &_matrix)
    : _data(_matrix), _next(nullptr), _prev(nullptr) {}

CircularQueueOfMatrixNode *CircularQueueOfMatrixNode::getPrev() {

```



```

    return _prev;
}

void CircularQueueOfMatrixNode::setPrev(CircularQueueOfMatrixNode *prev) {
    _prev = prev;
}

CircularQueueOfMatrixNode *CircularQueueOfMatrixNode::getNext() {
    return _next;
}

void CircularQueueOfMatrixNode::setNext(CircularQueueOfMatrixNode *next) {
    _next = next;
}

Matrix &CircularQueueOfMatrixNode::getMatrix() {
    return _data;
}

CircularQueueOfMatrix::CircularQueueOfMatrix() : _head(nullptr) {}

CircularQueueOfMatrix::~CircularQueueOfMatrix() {
    CircularQueueOfMatrixNode *iterator = _head, *next;
    do {
        if (!iterator)
            break;
        next = iterator->getNext();
        delete iterator;
        iterator = next;
    } while (iterator != _head);
}

void CircularQueueOfMatrix::add(Matrix &m) {
    CircularQueueOfMatrixNode *node = new CircularQueueOfMatrixNode(m);
    if (_head) {
        node->setNext(_head);
        node->setPrev(_head->getPrev());
        _head->getPrev()->setNext(node);
        _head->setPrev(node);
    } else {
        node->setNext(node);
        node->setPrev(node);
        _head = node;
    }
}

Matrix &CircularQueueOfMatrix::remove() {
    if (_head == nullptr)
        throw NotEnoughOfItems();
    Matrix &result = _head->getPrev()->getMatrix();
    CircularQueueOfMatrixNode *prev = _head->getPrev();
    if (_head == prev)
        _head = nullptr;
    else {
        prev->getPrev()->setNext(_head);
        _head->setPrev(prev->getPrev());
    }
}

```

```

    }
    delete prev;
    return result;
}

void CircularQueueOfMatrix::fprint(std::ostream &os) {
    int num = 0;
    CircularQueueOfMatrixNode *iterator = _head;
    do {
        if (!iterator) {
            os << "No items" << std::endl;
            return;
        }
        os << "(" << num++ << "): " << std::endl;
        iterator->getMatrix().fprint(os);
        iterator = iterator->getNext();
    } while (iterator != _head);
}

void CircularQueueOfMatrix::print() {
    fprint(std::cout);
}

const char *CircularQueueOfMatrix::NotEnoughOfItems::what() const noexcept {
    return "Not enough elements in matrix";
}

```

6.5. Выводы

В лабораторной работе было произведена реализация механизмов работы с исключениями – инструментами передачи управления вверх по стеку, с вызовом деструкторов объектов, расположенных на стеке. Этот механизм важен в разработке программных продуктов на языке c++, т.к. позволяет упростить код и являясь аналогом goto, быть безопасным из-за того, что исключения “разворачивают” стек вызывая деструкторы.

ЗАКЛЮЧЕНИЕ

В ходе выполнения лабораторных работ была изучена система пользовательских типов языка, перегрузка операторов, виртуальные функции, исключения. Все эти языковые конструкции являются частью реализации стандартного объектно-ориентированного подхода к разработке программного обеспечения на языке c++.