МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ  
РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное автономное образовательное учреждение

высшего образования   
**«Национальный исследовательский   
Нижегородский государственный университет им. Н.И. Лобачевского»**

**(ННГУ)**

**Институт информационных технологий, математики и механики**

Направление подготовки: «ПМИ»

**ОТЧЕТ**

по лабораторной работе

**ВЕКТОР**

**Выполнил:** студент группы

3822Б1ПМ2

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_Р.Д. Махнёв

Подпись

**Проверил:** лаборант ИИТММ

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Д.А. Черных

Подпись

Нижний Новгород  
2023

**Содержание**

1. [Введение 3](#_Toc131449590)
2. [Постановка задачи 3](#_Toc131449591)
3. [Описание алгоритмов 4](#_Toc131449592)
   1. [Ручной выбор ведущего элемента 4](#_Toc131449593)
   2. [Метод Жордана-Гаусса 4](#_Toc131449594)
   3. [Метод Гаусса 4](#_Toc131449595)
4. [Описание программы 5](#_Toc131449596)
5. [Результаты вычисления 12](#_Toc131449597)
6. [Заключение 13](#_Toc131449598)
7. [Приложения 15](#_Toc131449599)

# **Введение**

**Актуальность.** Вектор - это стандартный шаблон обобщённого программирования языка C++, реализующий динамический массив. [1] Векторы в C++ представляют собой контейнеры, которые могут хранить объекты любого типа. Они позволяют добавлять и удалять элементы в конец контейнера, а также получать доступ к элементам по индексу. [2]

Векторы в C++ могут быть использованы для хранения большого количества данных и обеспечивают быстрый доступ к элементам.

**Цель работы:** изучение работы вектора

**Постановка задачи**

1. изучить различные источники по данной теме;
2. реализовать класс вектор;
3. провести сравнение различных методов сортировки.

**Описание алгоритмов**

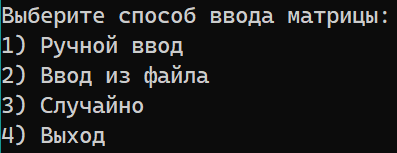
1. **Поля**
2. Указатель на область памяти, где будет находиться массив.
3. Размер вектора (count)
4. **Методы**
5. Обращение к элементу по индексу

**Описание программы**

Данная программа является консольным приложением, она может решать СЛАУ.

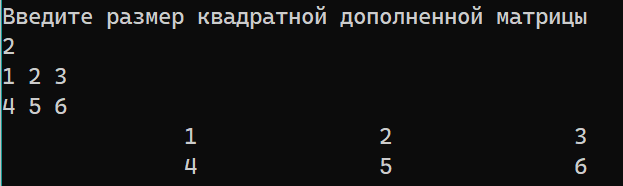
Программа получает на вход дополненную матрицу, записанную в файл, заданную пользователем вручную или с помощью генератора случайных чисел.

Изображение №1.



Выбор опций осуществляется вводом её номера.

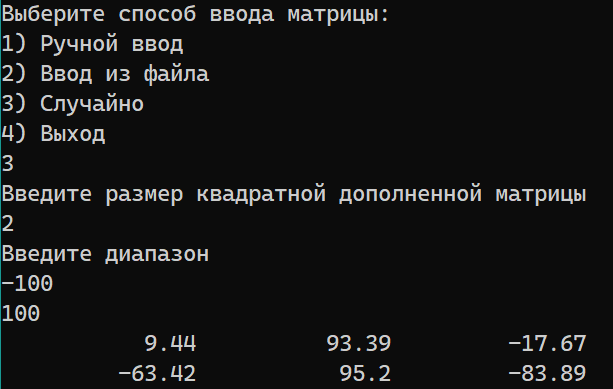
Изображение №2.



Ручной ввод коэффициентов осуществляется через пробел и перевода на новую строку.

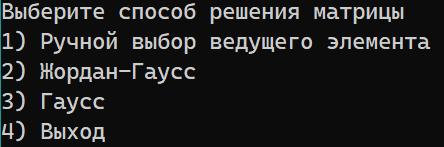
Ввод из файла осуществляется через файл input.txt, а вывод – output.txt.

Изображение №3.



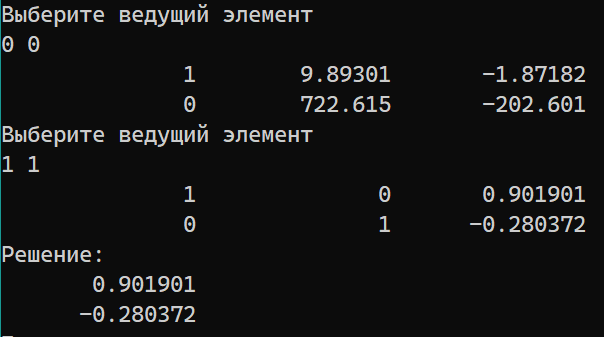
Для того, чтобы сгенерировать матрицу необходимо ввести размер матрицы n\*n, максимальные и минимальные принимаемые значения.

Изображение №4.



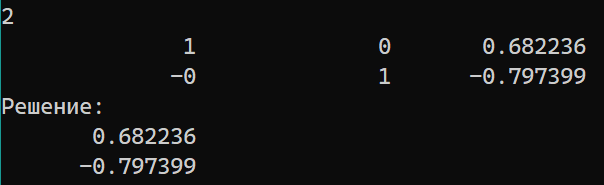
После ввода матрицы необходимо выбрать способ решения.

Изображение №5.



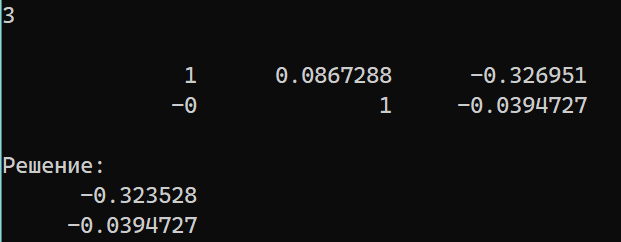
Во время ручного решения можно выбирать ведущий элемент. Каждый элемент выбирается из разных строк и столбцов.

Изображение №6.



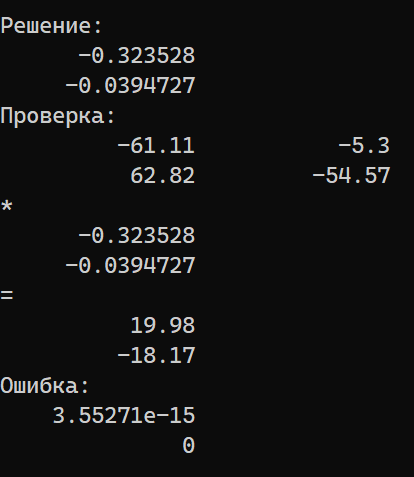
Решение методом Жордана-Гаусса.

Изображение №7.



Решение методом Гаусса.

Изображение №8.



Решение - вектор. Проверка представляет собой перемножение матрицы n\*n на вектор ответа.

Для выхода необходимо выбрать соответствующий пункт.

**Описание классов**

**class MatrixString –** содержит описание строки матрицы.

double\* string; - массив значений строки матрицы.

size\_t columns; - длина строки матрицы.

Конструкторы:

MatrixString();

MatrixString(const size\_t columns);

MatrixString(const MatrixString& string);

MatrixString(const double\* rationals, const size\_t columns);

Перегрузки операторов:

friend std::istream& operator>>(std::istream& stream, MatrixString& current);

friend std::ostream& operator<<(std::ostream& stream, const MatrixString& current);

friend MatrixString operator+ (const MatrixString& left, const MatrixString& right);

friend MatrixString operator- (const MatrixString& left, const MatrixString& right);

friend MatrixString operator\* (const MatrixString& left, double rationals);

friend MatrixString operator/ (const MatrixString& left, double rationals);

MatrixString& operator\*= (double rationals);

MatrixString& operator/= (double rationals);

double& operator[](size\_t i);

MatrixString& operator= (const MatrixString& equals);

Методы:

size\_t getColumnsCount(); - возвращает длину матричной строки

void fillRandom(int start, int end); - заполняет матричную строку случайными символами. Входные параметры: диапазон значений (start, end).

(Приложения №1, 8)

**class Matrix** –содержит описание матрицы.

MatrixString\* matrix; - массив матричных строк.

size\_t strings\_count; - количество строк в матрице.

Конструкторы:

Matrix(const size\_t strings, const size\_t columns);

Matrix(const MatrixString\* rationals, const size\_t strings, const size\_t columns);

Matrix(const Matrix& \_matrix);

Деструктор:

~Matrix();

Перегрузки операторов:

Matrix& operator= (const Matrix& equals);

MatrixString& operator[](size\_t i);

std::istream& operator>>(std::istream& stream, Matrix& current);

std::ostream& operator<<(std::ostream& stream, const Matrix& current);

Matrix operator+ (const Matrix& left, const Matrix& right);

Matrix operator\* (const Matrix& left, const Matrix& right);

Matrix operator- (const Matrix& left, const Matrix& right);

Методы:

void fillRandom(int start, int end); - заполняет матрицу случайными значениями. Входные параметры: диапазон значений (start, end).

int getStringsCount(); - возвращает количество строк в матрице.

void abs(); - находит модуль матрицы. (Приложение № 2, 7)

**class String** – содержит описание строки.

char\* content; - массив символов.

Конструкторы:

String(const char\* content);

String(const String& str);

String();

Деструктор:

~String();

Перегрузки операторов:

String& operator= (const String& str);

String operator+ (const String& right);

String& operator+= (const String& right);

friend std::ostream& operator<< (std::ostream& stream, const String& string);

Методы.

size\_t getLength() const; - возвращает количество значимых символов в строке.

void print(); - вывод строки. (Приложение №4, 9)

**class Exception** – содержит описание исключения.

String error\_name; - имя ошибки.

String file\_name; - имя файла.

int error\_code; - код ошибки.

Конструкторы:

Exception(const int error\_code, const String& error\_name, const String& file\_name);

Exception(const int error\_code, const char\* error\_name, const char\* file\_name);

Деструктор:

~Exception();

Перегрузки операторов:

std::ostream& operator<< (std::ostream& stream, const Exception exception);

Методы:

String getErrorName() const; - возвращает имя ошибки.

String getFileName() const; - возвращает имя файла. (Приложение №1, 5)

**Описание функций:**

Matrix nonAutoGauss(Matrix& added\_matrix, int n); - решение СЛАУ с выбором ведущего элемента. Входные параметры: added\_matrix – ссылка на дополненную матрицу, n – количество строк. Выходные данные: вектор ответа.

void JordanGauss(Matrix& matrix, size\_t string, size\_t column); - решение СЛАУ методом Жордана-Гаусса. Входные параметры: matrix – ссылка на дополненную матрицу, string – строка выбранного элемента, column – столбец выбранного элемента.

void ForwardGauss(Matrix& matrix); – прямой ход Гаусса. Приводит матрицу к треугольному виду. Входные параметры: matrix – ссылка на дополненную матрицу.

Matrix BackwardGauss(Matrix & matrix); – обратный ход Гаусса. Входные параметры: matrix – ссылка на дополненную матрицу. Выходные данные: вектор ответа.

Matrix getSolution(Matrix& added\_matrix, int n); - выбор способа решения. Возвращает матрицу ответа. Входные параметры: added\_matrix – ссылка на дополненную матрицу, n – количество строк. Выходные данные: вектор ответа.

Matrix Initialize(size\_t \_n); - заполнение дополненной матрицы. Входные параметры: n – количество строк. Выходные данные: исходная матрица.

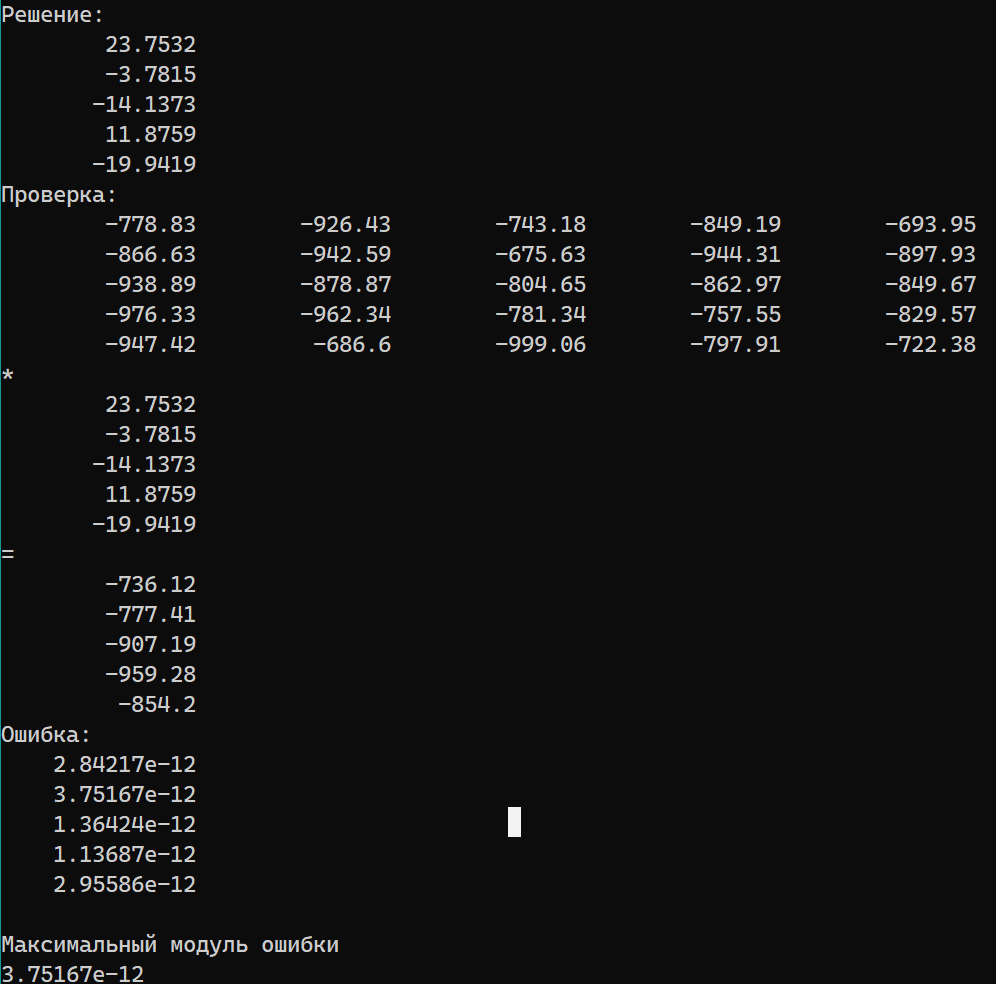
void Check(Matrix& matrix, size\_t string, size\_t column); - проверяет выбранный элемент на ноль. Если равен нулю, то просит сделать перестановку строк или выйти. Входные параметры: matrix – дополненная матрица, string – строка выбранного элемента, column – столбец выбранного элемента.

**Результаты вычисления**

Результатом вычисления является вектор решения СЛАУ, вектор ошибок и максимальная по модулю ошибка.

Изображение №9.

Решение СЛАУ с 5 неизвестными



Исходная матрица, приведённая матрица, вектор ошибок, вектор решений автоматически записываются в output.txt.

|  |  |  |  |
| --- | --- | --- | --- |
| Номер | Количество переменных | Диапазон | Максимальная ошибка |
| 1 | 5 | (-1000, 1000) | 3.75167e-12 |
| 2 | 100 | (-1000, 1000) | 2.76941e-10 |
| 3 | 1000 | (-1000, 1000) | 6.36433e-07 |
| 4 | 5000 | (-1000, 1000) | 6.93751e-06 |
| 5 | 2500 | (-10000, 10000) | 2.63396e-06 |
| 6 | 2500 | (-100000, 100000) | 1.35544e-06 |

**Заключение**

В этой работе были изучены и разработаны основные способы решения СЛАУ. По полученным данным можно сделать следующие выводы:

Программа достаточно точно находит решения СЛАУ. Чем больше переменных, тем ниже точность.

**Литература**

1. **Vector (C++) — Википедия**

<https://ru.wikipedia.org/wiki/Vector_(C%2B%2B)>

1. **C++ | Вектор – METANIT**

[**https://metanit.com/cpp/tutorial/7.2.php**](https://metanit.com/cpp/tutorial/7.2.php)

1. **3**

**Приложения**

Приложение №1

#pragma once

#include "String.hpp"

#include <ostream>

class Exception

{

public:

Exception(const int error\_code, const String& error\_name, const String& file\_name);

Exception(const int error\_code, const char\* error\_name, const char\* file\_name);

friend std::ostream& operator<< (std::ostream& stream, const Exception exception);

int getErrorCode() const;

String getErrorName() const;

String getFileName() const;

~Exception();

private:

String error\_name;

String file\_name;

int error\_code;

};

std::ostream& operator<< (std::ostream& stream, const Exception exception);

Приложение №2

#pragma once

#include "matrix\_string.hpp"

#include <istream>

class Matrix

{

public:

Matrix(const size\_t strings, const size\_t columns);

Matrix(const MatrixString\* rationals, const size\_t strings, const size\_t columns);

Matrix(const Matrix& \_matrix);

int getStringsCount();

void abs();

Matrix getVector(const size\_t num);

Matrix getSubMatrix(const size\_t string\_start, const size\_t column\_start, const size\_t string\_end, const size\_t column\_end);

void fillRandom(int start, int end);

Matrix& operator= (const Matrix& equals);

~Matrix();

MatrixString& operator[](size\_t i);

friend std::istream& operator>>(std::istream& stream, Matrix& current);

friend std::ostream& operator<<(std::ostream& stream, const Matrix& current);

friend Matrix operator+ (const Matrix& left, const Matrix& right);

friend Matrix operator\* (const Matrix& left, const Matrix& right);

friend Matrix operator- (const Matrix& left, const Matrix& right);

private:

MatrixString\* matrix;

size\_t strings\_count;

};

Matrix operator- (const Matrix& left, const Matrix& right);

std::istream& operator>>(std::istream& stream, Matrix& current);

std::ostream& operator<<(std::ostream& stream, const Matrix& current);

Matrix operator+ (const Matrix& left, const Matrix& right);

Matrix operator\* (const Matrix& left, const Matrix& right);

Matrix nonAutoGauss(Matrix& added\_matrix, int n);

void JordanGauss(Matrix& matrix, size\_t string, size\_t column);

Matrix getSolution(Matrix& added\_matrix, int n);

Приложение №3

#pragma once

#include "Exeption.hpp"

class MatrixString {

public:

MatrixString();

MatrixString(const size\_t columns);

MatrixString(const MatrixString& string);

MatrixString(const double\* rationals, const size\_t columns);

void fillRandom(int start, int end);

MatrixString& operator= (const MatrixString& equals);

~MatrixString();

double& operator[](size\_t i);

size\_t getColumnsCount();

friend std::istream& operator>>(std::istream& stream, MatrixString& current);

friend std::ostream& operator<<(std::ostream& stream, const MatrixString& current);

friend MatrixString operator+ (const MatrixString& left, const MatrixString& right);

friend MatrixString operator- (const MatrixString& left, const MatrixString& right);

friend MatrixString operator\* (const MatrixString& left, double rationals);

friend MatrixString operator/ (const MatrixString& left, double rationals);

MatrixString& operator\*= (double rationals);

MatrixString& operator/= (double rationals);

private:

double\* string;

size\_t columns;

};

Приложение №4

#pragma once

#include <ostream>

class String {

private:

char\* content;

public:

String(const char\* content);

String(const String& str);

String();

~String();

size\_t getLength() const;

void print();

String& operator= (const String& str);

String operator+ (const String& right);

String& operator+= (const String& right);

friend std::ostream& operator<< (std::ostream& stream, const String& string);

};

size\_t getLength(const char\* str);

std::ostream& operator<< (std::ostream& stream, const String& string);

Приложение №5

#include "Exeption.hpp"

Exception::Exception(const int error\_code, const String& error\_name, const String& file\_name)

{

this->error\_code = error\_code;

this->error\_name = error\_name;

this->file\_name = file\_name;

}

Exception::Exception(const int error\_code, const char\* error\_name, const char\* file\_name) {

this->error\_code = error\_code;

this->error\_name = String(error\_name);

this->file\_name = String(file\_name);

}

Exception::~Exception()

{

}

int Exception::getErrorCode() const {

return error\_code;

}

String Exception::getErrorName() const {

return error\_name;

}

String Exception::getFileName() const {

return file\_name;

}

std::ostream& operator<< (std::ostream& stream, const Exception exception) {

stream << "Error code: " << exception.error\_code << std::endl << "Error name: " << exception.error\_name << std::endl << "File name: " << exception.file\_name << std::endl;

return stream;

}

Приложение №6

#include <iostream>

#include <fstream>

#include <locale>

#include "matrix.hpp"

using std::cout;

using std::cin;

using std::endl;

Matrix Initialize(size\_t& \_n) {

int choice;

cin >> choice;

switch (choice)

{

case 1: {

cout << "Введите размер квадратной дополненной матрицы" << endl;

size\_t n;

cin >> n;

Matrix matrix(n, n + 1);

cin >> matrix;

\_n = n;

return matrix;

break;

}

case 2: {

std::ifstream file("input.txt");

size\_t n;

file >> n;

Matrix matrix(n, n + 1);

\_n = n;

file >> matrix;

return matrix;

break;

}

case 3: {

cout << "Введите размер квадратной дополненной матрицы" << endl;

size\_t n;

cin >> n;

Matrix matrix(n, n + 1);

cout << "Введите диапазон" << endl;

int start, end;

cin >> start >> end;

if (start > end) {

int current = start;

start = end;

end = current;

}

\_n = n;

matrix.fillRandom(start, end);

return matrix;

break;

}

case 4:

std::exit(0);

break;

default:

throw Exception(4, "Choice error", "main.cpp");

break;

}

}

int main() try{

setlocale(LC\_ALL, "Russian");

srand(time(NULL));

while (true) {

cout << "Выберите способ ввода матрицы:" << endl << "1) Ручной ввод" << endl << "2) Ввод из файла" << endl << "3) Случайно" << endl << "4) Выход" << endl;

size\_t n;

Matrix added\_matrix = Initialize(n);

if (n < 10) {

cout << added\_matrix;

}

Matrix copy\_matrix = added\_matrix;

cout << "Выберите способ решения матрицы" << endl << "1) Ручной выбор ведущего элемента" << endl << "2) Жордан-Гаусс" << endl << "3) Гаусс" << endl << "4) Выход" << endl;

Matrix solution = getSolution(added\_matrix, n);

cout << "Решение: " << endl << solution;

Matrix submatrix = copy\_matrix.getSubMatrix(0, 0, n, n);

if (n < 10) {

cout << "Проверка:" << endl << submatrix << "\*" << endl << solution << "=" << endl;

}

Matrix multiple = submatrix \* solution;

cout << multiple;

Matrix error = multiple - copy\_matrix.getVector(n);

error.abs();

cout << "Ошибка: " << endl << error;

std::ofstream output("output.txt");

output << "Дано" << endl << copy\_matrix << endl << "Решение" << endl << added\_matrix << endl << "Ответ" << endl << solution << "Ошибка" << endl << error;

double max\_error = 0;

for (size\_t i = 0; i < n; i++)

{

double current = abs(error[i][0]);

if (current > max\_error) {

max\_error = current;

}

}

cout << endl << "Максимальный модуль ошибки"<< endl << max\_error << endl;

output << endl<< "Максимальный модуль ошибки" << endl << max\_error;

}

}

catch (Exception error) {

std::cout << error;

}

catch (...) {

std::cout << "Something went wrong";

}

Приложение №7

#include "matrix.hpp"

#include <iomanip>

#include <ctime>

#include <iostream>

#include <cstdlib>

#define EPSILON 1e-7

Matrix::Matrix(const size\_t strings, const size\_t columns) {

if ((strings <= 0) || (columns <= 0))

throw Exception(3, "Constructor error", "matrix.cpp");

matrix = new MatrixString[strings];

for (size\_t i = 0; i < strings; i++)

{

matrix[i] = MatrixString(columns);

}

strings\_count = strings;

}

Matrix::Matrix(const MatrixString\* rationals,const size\_t strings, const size\_t columns)

{

if ((strings <= 0) || (columns <= 0))

throw Exception(3, "Constructor error", "matrix.cpp");

matrix = new MatrixString[strings];

for (size\_t i = 0; i < strings; i++)

{

matrix[i] = rationals[i];

}

strings\_count = strings;

}

Matrix::Matrix(const Matrix& \_matrix)

{

matrix = new MatrixString [\_matrix.strings\_count];

for (size\_t i = 0; i < \_matrix.strings\_count; i++)

{

matrix[i] = \_matrix.matrix[i];

}

strings\_count = \_matrix.strings\_count;

}

Matrix::~Matrix(){

delete[] matrix;

strings\_count = 0;

}

MatrixString& Matrix::operator[](size\_t i)

{

return matrix[i];

}

Matrix operator-(const Matrix& left, const Matrix& right)

{

if ((left.strings\_count != right.strings\_count) || (left.matrix[0].getColumnsCount() != right.matrix[0].getColumnsCount())) {

throw Exception(3, "Matrix summary error", "matrix.cpp");

}

Matrix summary(left.strings\_count, left.matrix[0].getColumnsCount());

for (size\_t i = 0; i < left.strings\_count; i++)

{

summary.matrix[i] = left.matrix[i];

summary.matrix[i] = summary.matrix[i] - right.matrix[i];

}

return summary;

}

std::istream& operator>>(std::istream& stream, Matrix& current) {

for (size\_t i = 0; i < current.strings\_count; i++)

{

stream >> current.matrix[i];

}

return stream;

}

std::ostream& operator<<(std::ostream& stream, const Matrix& current) {

for (size\_t i = 0; i < current.strings\_count; i++)

{

stream << current.matrix[i];

stream << std::endl;

}

return stream;

}

Matrix operator+(const Matrix& left, const Matrix& right)

{

if ((left.strings\_count != right.strings\_count) || (left.matrix[0].getColumnsCount() != right.matrix[0].getColumnsCount())) {

throw Exception(3, "Matrix summary error", "matrix.cpp");

}

Matrix summary(left.strings\_count, left.matrix[0].getColumnsCount());

for (size\_t i = 0; i < left.strings\_count; i++)

{

summary.matrix[i] = left.matrix[i];

summary.matrix[i] = summary.matrix[i] + right.matrix[i];

}

return summary;

}

Matrix operator\*(const Matrix& left, const Matrix& right)

{

if (left.matrix[0].getColumnsCount() != right.strings\_count) throw Exception(6, "Multiple matrix error", "matrix.cpp");

size\_t columns\_count = right.matrix[0].getColumnsCount();

Matrix multiple(left.strings\_count, columns\_count);

for (size\_t i = 0; i < left.strings\_count; i++)

{

size\_t cur\_columns\_count = left.matrix[0].getColumnsCount();

for (size\_t k = 0; k < columns\_count; k++)

{

double sum = 0;

for (size\_t j = 0; j < cur\_columns\_count; j++)

{

sum += left.matrix[i][j] \* right.matrix[j][k];

}

multiple.matrix[i][k] = sum;

}

}

return multiple;

}

void Matrix::fillRandom(int start, int end) {

for (size\_t i = 0; i < strings\_count; i++)

{

matrix[i].fillRandom(start, end);

}

}

int Matrix::getStringsCount()

{

return strings\_count;

}

void Matrix::abs()

{

using std::abs;

if (this->matrix == nullptr)

throw Exception(15, "abs of null", "matrix.hpp");

size\_t columns = matrix[0].getColumnsCount();

for (size\_t i = 0; i < this->strings\_count; i++)

{

for (size\_t j = 0; j < columns; j++)

{

this->matrix[i][j] = abs(matrix[i][j]);

}

}

}

Matrix Matrix::getVector(const size\_t num)

{

Matrix vector(this->strings\_count, 1);

size\_t count = vector.strings\_count;

for (size\_t i = 0; i < count; i++)

{

vector.matrix[i][0] = this->matrix[i][num];

}

return vector;

}

Matrix Matrix::getSubMatrix(const size\_t string\_start, const size\_t column\_start, const size\_t string\_end, const size\_t column\_end)

{

if (this->matrix == nullptr) throw Exception(10, "you can not get submatrix from nullptr", "matrix.cpp");

if (column\_start > column\_end) throw Exception(10, "submatrix column error", "matrix.cpp");

if (string\_start > string\_end) throw Exception(10, "submatrix string error", "matrix.cpp");

if (column\_start > this->matrix[0].getColumnsCount()) throw Exception(11, "submatrix incorrect index", "matrix.cpp");

Matrix sub\_matrix((string\_end - string\_start), (column\_end - column\_start));

for (size\_t i = string\_start; i < string\_end; i++)

{

for (size\_t j = column\_start; j < string\_end; j++)

{

sub\_matrix.matrix[i - string\_start][j] = this->matrix[i][j];

}

}

return sub\_matrix;

}

Matrix& Matrix::operator=(const Matrix& equals)

{

if (this->matrix != nullptr) {

delete[] matrix;

this->strings\_count = 0;

}

this->matrix = new MatrixString[equals.strings\_count];

for (size\_t i = 0; i < equals.strings\_count; i++)

{

this->matrix[i] = equals.matrix[i];

}

this->strings\_count = equals.strings\_count;

return \*this;

}

void Check(Matrix& matrix, size\_t string, size\_t column) {

if (abs(matrix[string][column]) <= EPSILON) {

std::cout << "Ошибка выберите действие:" << std::endl << "1) пеереставить строчки" << std::endl << "2) закончить работу" << std::endl;

std::cout << matrix;

int choice;

std::cin >> choice;

switch (choice)

{

case 1:

{

size\_t from, to;

std::cout << "переставить строчки откуда куда";

std::cin >> from >> to;

MatrixString copy = matrix[from];

matrix[from] = matrix[to];

matrix[to] = copy;

}

case 2:

{

std::exit(0);

}

default:

throw Exception(9, "transform error", "matrix.cpp");

break;

}

}

}

void JordanGauss(Matrix& matrix, size\_t string, size\_t column) {

Check(matrix, string, column);

matrix[string] /= matrix[string][column];

for (size\_t i = 0; i < string; i++)

{

matrix[i] = matrix[i] - matrix[string] \* matrix[i][column];

}

for (size\_t i = string + 1; i < matrix.getStringsCount(); i++)

{

MatrixString cur = (matrix[string] \* matrix[i][column]);

matrix[i] = matrix[i] - cur;

}

}

void ForwardGauss(Matrix& matrix) {

size\_t n = matrix.getStringsCount();

for (size\_t i = 0; i < n; i++)

{

size\_t j;

for (j = 0; j < n; j++)

{

if (abs(matrix[i][j]) >= EPSILON)

break;

}

if (j >= n) {

if ((abs(matrix[i][n - 1]) <= EPSILON) && (abs(matrix[i][n]) <= EPSILON))

throw Exception(8, "Данная СЛАУ имеет много решений", "matrix.cpp");

else

{

throw Exception(8, "Данная СЛАУ не имеет решений", "matrix.cpp");

}

}

Check(matrix, i, i);

matrix[i] /= matrix[i][i];

for (size\_t k = i + 1; k < n; k++)

{

matrix[k] = matrix[k] - matrix[i] \* matrix[k][i];

}

}

if (n < 10) {

std::cout << std::endl << matrix << std::endl;

}

}

Matrix BackwardGauss(Matrix & matrix) {

size\_t n = matrix.getStringsCount();

Matrix result(n, 1);

result[n - 1][0] = matrix[n - 1][n];

for (long long int i = n - 2; i >= 0; i--)

{

double sum = 0;

for (size\_t j = i + 1; j < n; j++)

{

sum += matrix[i][j] \* result[j][0];

}

result[i][0] = (matrix[i][n] - sum);

}

return result;

}

Matrix nonAutoGauss(Matrix& added\_matrix, int n) {

bool\* choice\_strings = new bool[n];

bool\* choice\_column = new bool[n];

size\_t\* strings = new size\_t[n];

for (size\_t i = 0; i < n; i++)

{

choice\_strings[i] = false;

choice\_column[i] = false;

strings[i] = 0;

}

for (size\_t i = 0; i < n; i++)

{

size\_t j;

for (j = 0; j < n; j++)

{

if (abs(added\_matrix[i][j]) >= EPSILON)

break;

}

if (j >= n) {

if ((abs(added\_matrix[i][n - 1]) <= EPSILON) && (abs(added\_matrix[i][n]) <= EPSILON))

throw Exception(8, "Данная СЛАУ имеет много решений", "matrix.cpp");

else

{

throw Exception(8, "Данная СЛАУ не имеет решений", "matrix.cpp");

}

}

size\_t string, column;

std::cout << "Выберите ведущий элемент" << std::endl;

std::cin >> string >> column;

if (string > added\_matrix.getStringsCount()) {

delete[] choice\_strings;

delete[] choice\_column;

delete[] strings;

throw Exception(5, "incorrect element", "matrix.cpp");

}

if (column > added\_matrix[0].getColumnsCount()) {

delete[] choice\_strings;

delete[] choice\_column;

delete[] strings;

throw Exception(5, "incorrect element", "matrix.cpp");

}

while (choice\_strings[string]) {

std::cout << "Выберите ведущий элемент" << std::endl;

std::cin >> string >> column;

}

while (choice\_column[column]) {

std::cout << "Выберите ведущий элемент" << std::endl;

std::cin >> string >> column;

}

JordanGauss(added\_matrix, string, column);

strings[i] = string;

choice\_strings[string] = true;

choice\_column[column] = true;

if (n < 10) {

std::cout << added\_matrix;

}

}

Matrix solution(added\_matrix.getStringsCount(), 1);

for (size\_t i = 0; i < n; i++)

{

solution[strings[i]][0] = added\_matrix[i][n];

}

delete[] choice\_strings;

delete[] choice\_column;

delete[] strings;

return solution;

}

Matrix getSolution(Matrix& added\_matrix, int n) {

size\_t choice;

std::cin >> choice;

switch (choice)

{

case 1:

return nonAutoGauss(added\_matrix, n);

break;

case 2: {

for (size\_t i = 0; i < n; i++)

{

size\_t j;

for (j = 0; j < n; j++)

{

if (abs(added\_matrix[i][j])>= EPSILON)

break;

}

if (j < n) {

JordanGauss(added\_matrix, i, i);

}

else

{

if ((abs(added\_matrix[i][n - 1]) <= EPSILON) &&(abs(added\_matrix[i][n]) <= EPSILON))

throw Exception(8, "Данная СЛАУ имеет много решений", "matrix.cpp");

else

{

throw Exception(8, "Данная СЛАУ не имеет решений", "matrix.cpp");

}

}

}

}

break;

case 3:

ForwardGauss(added\_matrix);

return BackwardGauss(added\_matrix);

break;

case 4:

std::exit(0);

break;

default:

throw Exception(1, "incorrect input", "matrix.cpp");

break;

}

if (n < 10) {

std::cout << added\_matrix;

}

return added\_matrix.getVector(n);

}

Приложение №8

#include "matrix\_string.hpp"

#include "Exeption.hpp"

#include <iomanip>

std::istream& operator>>(std::istream& stream, MatrixString& current)

{

for (size\_t i = 0; i < current.columns; i++)

{

stream >> current.string[i];

}

return stream;

}

std::ostream& operator<<(std::ostream& stream, const MatrixString& current)

{

for (size\_t i = 0; i < current.columns; i++)

{

stream << std::setw(15) << current.string[i];

}

return stream;

}

MatrixString operator+(const MatrixString& left, const MatrixString& right)

{

if (left.columns != right.columns) throw Exception(3, "Column numbers are not equals", "matrix\_string.cpp");

MatrixString summary(left.columns);

for (size\_t i = 0; i < left.columns; i++)

{

summary.string[i] = right.string[i] + left.string[i];

}

return summary;

}

MatrixString operator-(const MatrixString& left, const MatrixString& right)

{

if (left.columns != right.columns) throw Exception(3, "Column numbers are not equals", "matrix\_string.cpp");

MatrixString minus(left.columns);

for (size\_t i = 0; i < left.columns; i++)

{

minus.string[i] = left.string[i] - right.string[i];

}

return minus;

}

MatrixString operator\*(const MatrixString& left, double rationals)

{

MatrixString multiple(left.columns);

for (size\_t i = 0; i < left.columns; i++)

{

multiple.string[i] = left.string[i] \* rationals;

}

return multiple;

}

MatrixString operator/(const MatrixString& left, double rationals)

{

MatrixString del(left.columns);

for (size\_t i = 0; i < left.columns; i++)

{

del.string[i] = left.string[i] / rationals;

}

return del;

}

MatrixString::MatrixString()

{

columns = 0;

string = nullptr;

}

MatrixString::MatrixString(const size\_t columns)

{

string = new double[columns];

this->columns = columns;

}

MatrixString::MatrixString(const MatrixString& string)

{

this->string = new double(string.columns);

for (size\_t i = 0; i < string.columns; i++)

{

this->string[i] = string.string[i];

}

this->columns = string.columns;

}

MatrixString::MatrixString(const double\* rationals, const size\_t columns)

{

string = new double[columns];

for (size\_t i = 0; i < columns; i++)

{

string[i] = rationals[i];

}

this->columns = columns;

}

void MatrixString::fillRandom(int start, int end)

{

start \*= 100;

end \*= 100;

for (size\_t i = 0; i < columns; i++)

{

string[i] = (rand() % (end - start + 1) + start) / 100.0;

}

}

MatrixString& MatrixString::operator=(const MatrixString& equals)

{

//std::cout << equals << std::endl;

if (string != nullptr) {

delete[] string;

string = nullptr;

columns = 0;

}

string = new double[equals.columns];

for (size\_t i = 0; i < equals.columns; i++)

{

string[i] = equals.string[i];

}

columns = equals.columns;

/\* std::cout << (\*this) << std::endl;\*/

return \*this;

}

MatrixString::~MatrixString()

{

if (string != nullptr) {

delete[] string;

}

columns = 0;

}

double& MatrixString::operator[](size\_t i)

{

return string[i];

}

size\_t MatrixString::getColumnsCount()

{

return columns;

}

MatrixString& MatrixString::operator\*=(double rationals)

{

for (size\_t i = 0; i < columns; i++)

{

string[i] = string[i] \* rationals;

}

return \*this;

}

MatrixString& MatrixString::operator/=(double rationals)

{

for (size\_t i = 0; i < columns; i++)

{

string[i] = string[i] / rationals;

}

return \*this;

}

Приложение №9

#include <iostream>

#include "String.hpp"

size\_t getLength(const char\* str) {

size\_t len = 0;

while (str[len] != '\0') {

len++;

}

return len;

}

String::String(const char\* str) {

/\*std::cout << "using constructor const char\*" << std::endl;\*/

using::getLength;

if (str != nullptr) {

size\_t length = getLength(str);

content = new char[length + 1];

for (size\_t i = 0; i < length; i++)

{

content[i] = str[i];

}

content[length] = '\0';

}

else

{

content = nullptr;

}

}

String::String(const String& str) : String(str.content) {

}

String::String() : String(nullptr) {

};

String::~String() {

if (content != nullptr) {

delete[] content;

content = nullptr;

}

}

void String::print() {

size\_t i = 0;

while (content[i] != '\0') {

std::cout << content[i];

}

}

size\_t String::getLength() const {

using::getLength;

return getLength(content);

}

String& String::operator=(const String& str) {

if (this->content != str.content) {

if (content != nullptr) {

delete[] content;

content = nullptr;

}

size\_t length = str.getLength();

content = new char[length + 1];

for (size\_t i = 0; i < length; i++)

{

content[i] = str.content[i];

}

content[length] = '\0';

}

return \*this;

}

String String::operator+ ( const String& right) {

size\_t left\_length = (\*this).getLength();

size\_t right\_length = right.getLength();

size\_t summary\_length = left\_length + right\_length;

char\* current = new char[summary\_length + 1];

size\_t i;

for (i = 0; i < left\_length; i++)

{

current[i] = this->content[i];

}

for (i = 0; i < summary\_length; i++)

{

current[i] = right.content[i - right\_length];

}

current[summary\_length] = '\0';

return String(current);

}

String& String::operator+= (const String& right) {

\*this = (\*this) + right;

return \*this;

}

std::ostream& operator<< (std::ostream& stream, const String& string) {

stream << string.content;

return stream;

}