МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ

РОССИЙСКОЙ ФЕДЕРАЦИИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ

ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

«ВОРОНЕЖСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»

#### Факультет экономики, менеджмента и информационных технологий

(факультет)

### Кафедра Систем управления и информационных технологий в строительств

# ОТЧЁТ ПО ЛАБОРАТОРНОЙ РАБОТЕ № 3

по дисциплине Объектно-ориентированное программирование

Тема «Подпрограммы»

Выполнил студент группы ИСТ-214 Д. В. Тюленев Подпись, дата Инициалы, фамилия

## Принял Е. Н. Королев Подпись, дата Инициалы, фамилия

Защищена \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Оценка \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

дата

2022

Лабораторная работа № 3

«Подпрограммы»

Вариант №4

1. Цель работы: Научиться работать с подпрограммами на языке С++
2. Задания на лабораторную работу: Написать функцию для заполнения квадратной матрицы размерностью n x n возрастающей последовательностью чисел по заданной схеме. Размерность матрицы задаётся константой и должна находиться в пределах 4 ≤ n ≤ 16. Написать функции вывода матрицы на экран и функцию нахождения обратной матрицы, функцию умножения матриц. Показать, что найдена обратная матрица, перемножить её с исходной.

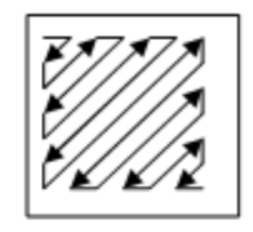


Рисунок 1 － Заданная схема варианта

1. Ход выполнения:

Для начала определяется новый тип структуры «lab3::SquareMatrix», которая представляет шаблон квадратной матрицы. Внутри тела структуры помещается конструктор c публичным модификатором доступа, который принимает в качестве параметра «size» － размер матрицы; он собирает матрицу из объекта «vector» необходимого размера при помощи метода «vector::push\_back», и устанавливает все клетки нулевыми значениями. Структура содержит закрытое поле «lab3::SquareMatrix::matrix» типа «matrix\_t» (псевдоним типа vector<vector<double>>) , которое хранит в себе двумерный массив описывающий матрицу.

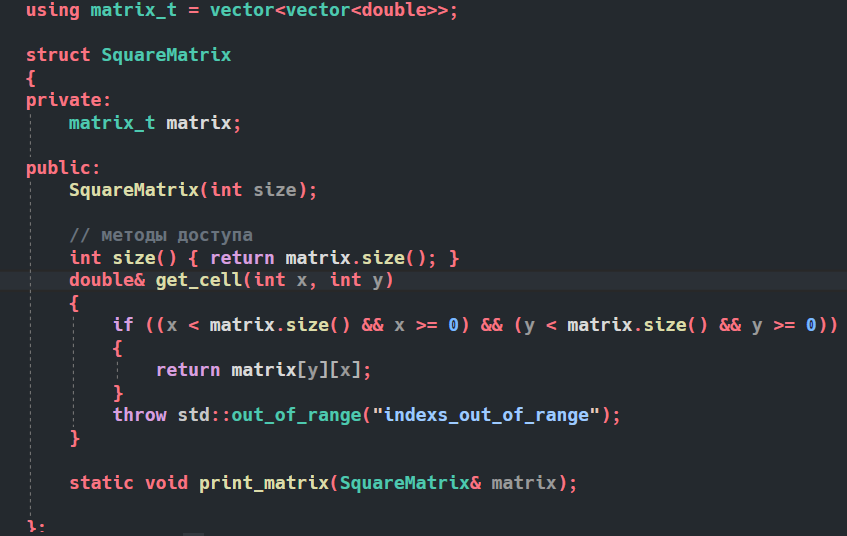


Рисунок 2 － Структура квадратной матрицы

Также определены два метода доступа: «lab3::SquareMatrix::size» － возвращает размер матрицы, «lab3::SquareMatrix::get\_cell» － принимает в качестве параметров местоположение выбранной клетки относительно матрицы и, если значения верные, то возвращает ссылку на выбранный элемент, иначе будет сгенерировано исключение. Для вывода квадратной матрицы на экран определён статический метод «lab3::SquareMatrix::print\_matrix», который принимает ссылку для печати необходимого объекта типа «lab3::SquareMatrix».

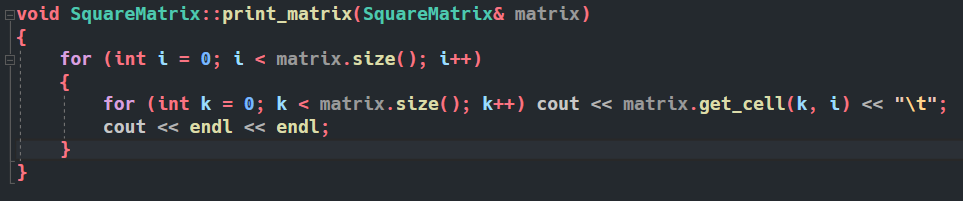


Рисунок 3 － Метод вывода матрицы на экран

Для заполнения квадратной матрицы определена функция «lab3::fill\_matrix». Она принимает в качестве параметра «matrix» － ссылку на объект незаполненной квадратной матрицы. В теле функции инициализируется переменная для счёта последовательности, устанавливается значение для клетки с координатами (х = 0; у = 0) и создаётся объект перечисления, отвечающего за направление движения вдоль стены (вниз или вправо).

Далее открывается цикл с условие пока курсор не будет находиться в в правом нижнем углу. В теле цикла совершается проверка на возможность совершения движения: если справа или снизу если место и текущее направление движения вдоль стены совпадает, то совершается перемещение курсора, установка значения для новой клетки и переключение движения; иначе, если лишь с одной из сторон свободно для перемещения, но направление движения не сходится, то происходит смена направления движения и возврат к началу текущей итерации.

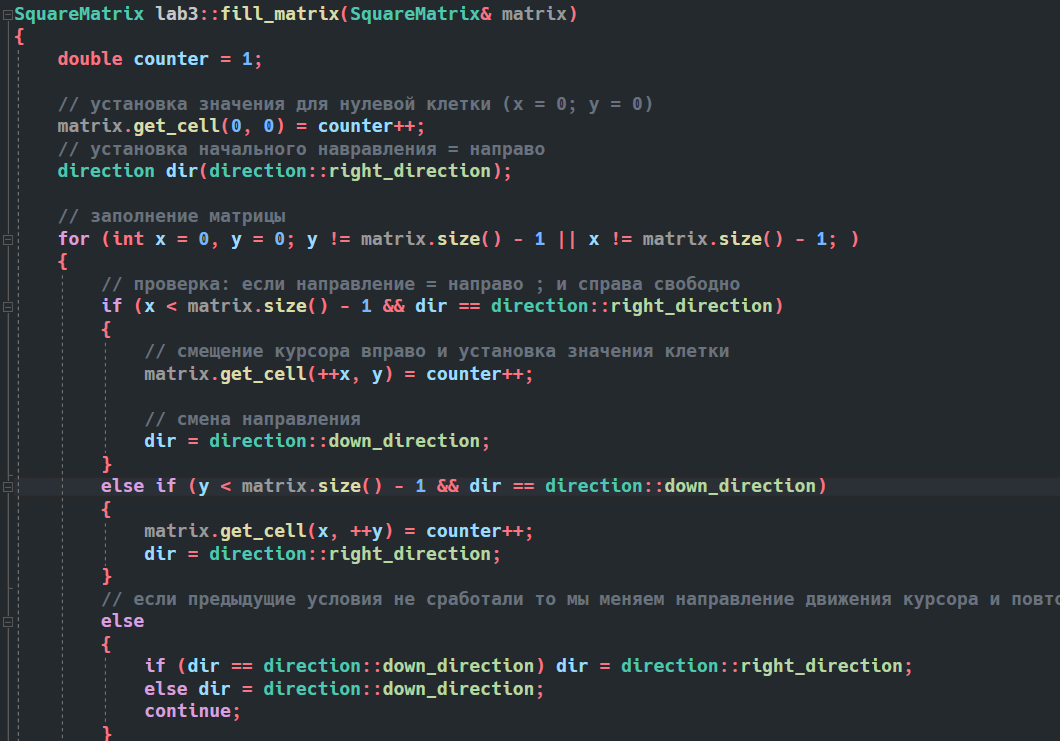


Рисунок 4 － Функция заполнения матрицы

После смещения вдоль стены производится проверка около каких стенок в данный момент курсор: если около верхней или вправо стены, то необходимо двигаться по диагонали влево и вниз; иначе необходимо двигаться вверх и вправо. Для осуществления движения по диагонали применяется цикл «while» c установленным условием двигать курсор и устанавливать значения клеток, пока он (курсор) не достигнет противоположной пары стенок.

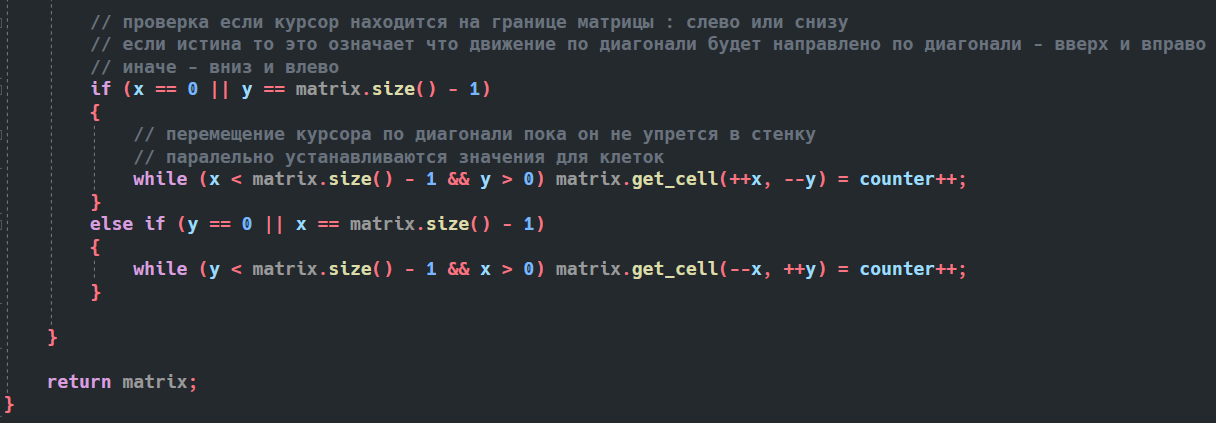


Рисунок 5 － Функция заполнения матрицы (продолжение)

После того, как курсор дошёл до конца и установил значения для всех клеток, то с помощью оператора «return »возвращается ссылка на заполненную матрицу.

Для нахождения обратной матрицы необходимо понять является ли исходная матрица валидной (невырожденной), для этого необходимо вычислить определитель матрицы и сравнить с нулём: если значение отлично от нуля, то для матрицы существует обратная; иначе нет.

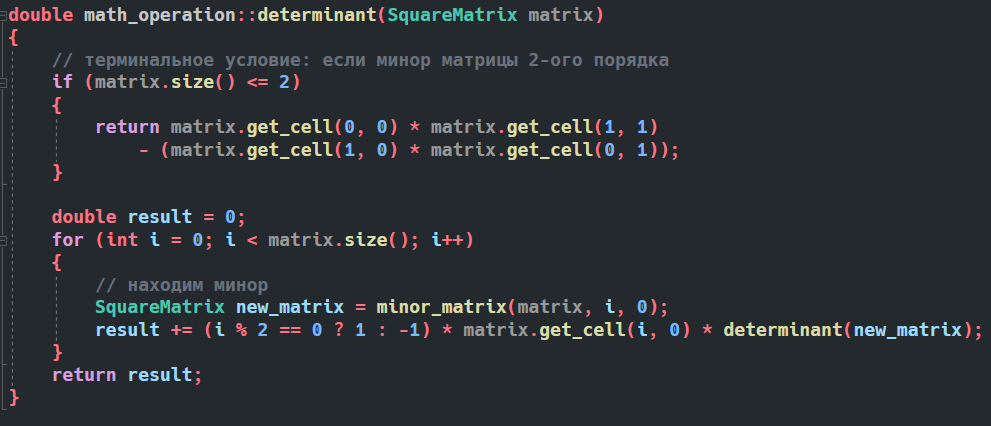


Рисунок 6 － Функция нахождения определителя

Определяется функция «lab3::math\_operation::determinant», которая принимает объект типа «lab3::SquareMatrix» и возвращает результат посчитанного определителя вещественного типа. В теле функции для начала определим терминальное условие: если минор матрицы меньше или равен второму порядку, то возвращаем посчитанное выражение:

Далее инициализируем переменную для расчёта результата определителя и открываем цикл «for» с диапазоном от «0» до «matrix.size()» (размер текущей матрицы). В теле цикла при каждой итерации находим минор текущей матрицы путём вычёркивания каждой клетки первой строки. Далее в переменную «result» при каждой итерации внутреннего цикла добавляется значение равное:

Расчёт величины происходит через рекурсивный вызов используя в качестве аргумента результат выполнения функции «lab3::math\_operation::minor\_matrix». После через оператор «return» возвращается значение определителя.

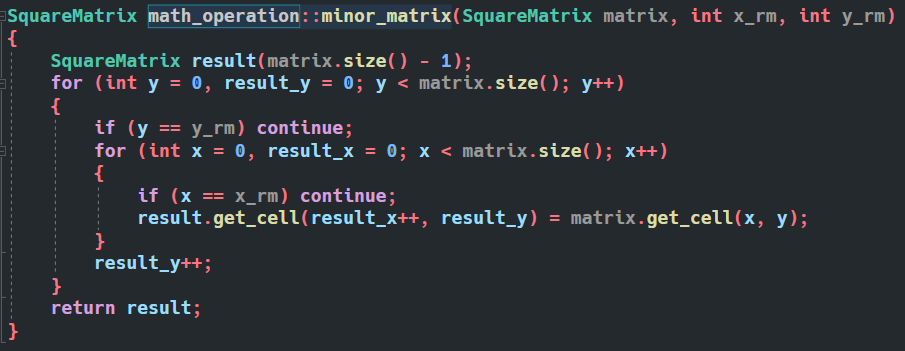


Рисунок 7 － Функция составления минора

Функция «lab3::math\_operation::minor\_matrix» принимает в качестве параметров: «matrix» － исходная матрица, «x\_rm» －столбец для вычёркивания, «y\_rm» －строка для вычёркивания; возвращает составленную матрицу уменьшенного нового порядка (n - 1).

В теле функции инициализируется новый объект квадратной матрицы с уменьшенным по сравнению с текущей размером (уменьшенным порядком). Через циклы «for» происходит установка значений для новой матрицы, путём игнорирования вычеркнутых ячеек исходной матрицы с помощью оператора «continue». После составленная матрица возвращается из функции.

Для нахождения обратной матрицы определена функция «lab3::math\_operation::inverse\_matrix». Она принимает в качестве параметра «matrix» － исходная матрица, и возвращает указатель на составленную обратную матрицу.

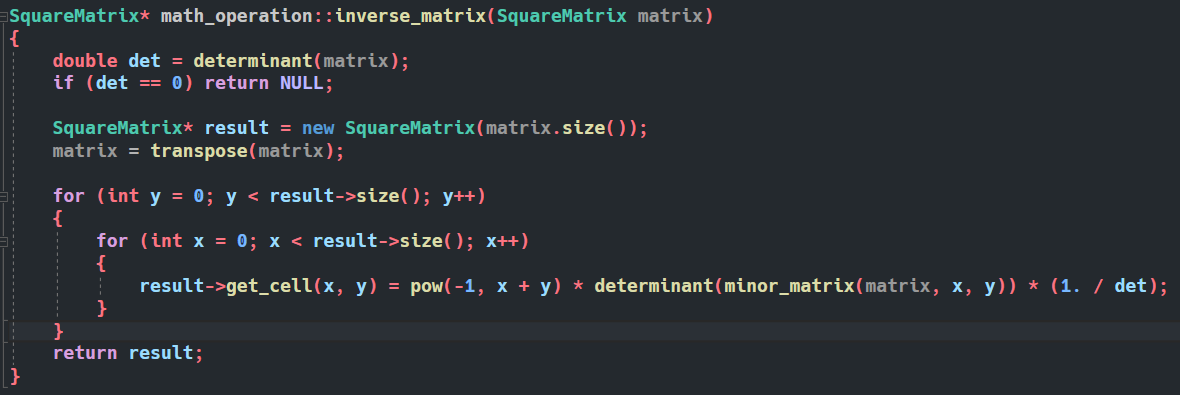


Рисунок 8 － Функция нахождения обратной матрицы

В начале проверяется можно ли из исходной матрицы составить обратную. Для этого при помощи вызова функции «lab3::math\_operation::determinant» вычисляется определитель и после сравнивается с нулём: если они равны, то возвращаем пустой указатель.

Далее выделяем память для хранения результата функции, т.е. обратной матрицы, и транспонируем исходную матрицу при помощи метода «lab3::math\_operation::transpose».

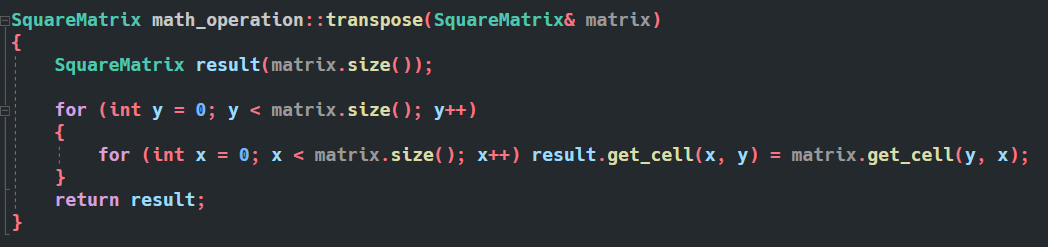


Рисунок 9 － Функция нахождения транспонированной матрицы

Данная функция принимает в качестве параметра ссылку на объект «lab3::SquareMatrix» представляющий исходную матрицу и возвращает транспонированную. С помощью двух вложенных циклов «for» на позиции строк в новой матрице устанавливаются столбцы исходной.

После в функции для расчёта обратной матрицы открываются два цикла для перемещения курсора по всем ячейкам новой созданной матрицы, и на его позицию устанавливается значение выражения: . После всего значение возвращается из функции.

После определяется функция «lab3::math\_operation::multiply\_matrix» для перемножения двух матриц. Данная функция в качестве параметра принимает два объекта типа «lab3::SquareMatrix» и возвращает указатель на результат произведения.

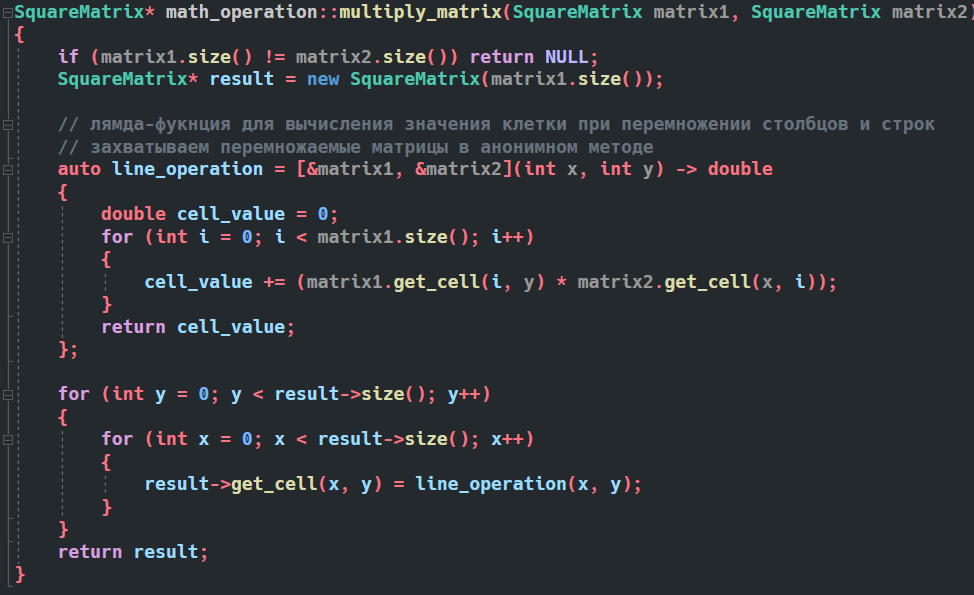


Рисунок 10 － Функция перемножения двух квадратных матриц

В начале совершается проверка на соответствие размеров двух матриц: если размеры разные возвращается пустой указатель. После выделяется память для хранения результата вычисления.

Далее инициализируется анонимная функция «line\_operation» для расчёта значения клетки результирующей матрицы. Лямбда захватывает из доступной области видимости две матрицы, определяет два параметра: координаты x и y; и возвращает вещественный тип. В теле анонимной функции инициализируется переменная хранящая результат. В цикле «for» со значением от «0» до «matrix1.size()» производятся перемножения строк и столбцов двух матриц и добавление в переменную «result». После лямбды открывается два вложенных цикла «for» для перемещения курсора по клеткам новой матрицы и записи результата вызова «line\_operation».

Далее в основной функции «lab3::run» определяется инициализируется переменная для хранения значения размера матрицы. После циклируется опрос пользователя в консоли, пока не будет введено валидное значение.

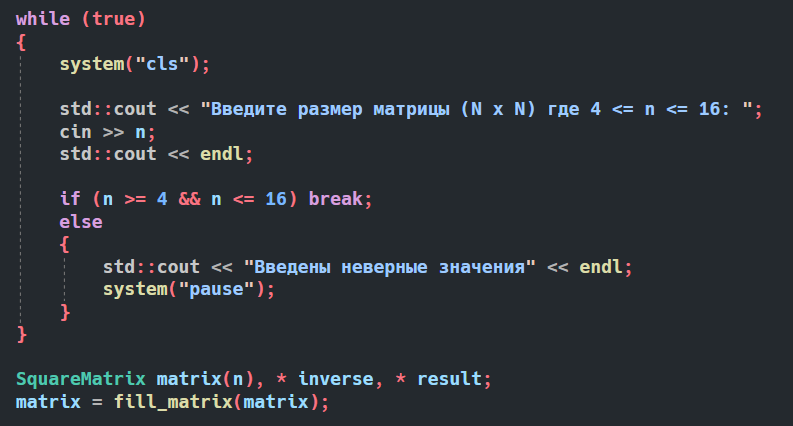


Рисунок 11 － Решение задания

После этого создаются экземпляры объекта «lab3::SquareMatrix» для хранения сгенерированной, обратной и перемноженной матриц. После поочерёдно производятся вызовы соответствующих функций с промежуточной печатью на экран значений матриц при помощи метода «lab3::SquareMatrix::print\_matrix».

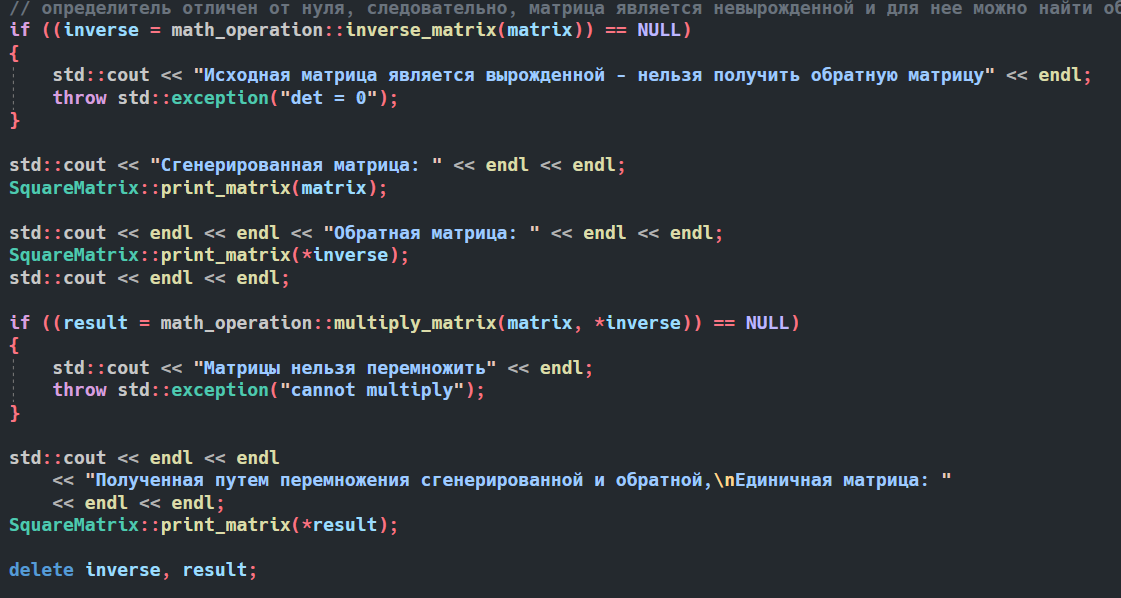


Рисунок 12 － Решение задания (продолжение)



Рисунок 13 － Вывод решения (часть 1)

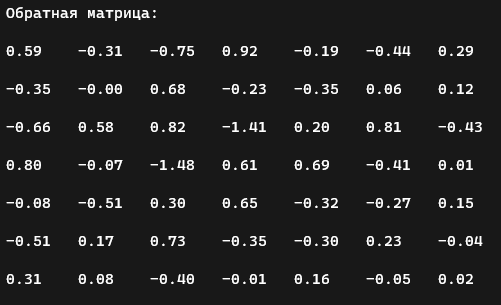


Рисунок 13 － Вывод решения (часть 2)



Рисунок 13 － Вывод решения (часть 3)

Вывод: Научились работать с подпрограммами на языке С++.

Приложение А

Листинг программного кода языка C++

Файл «lab2.h»

#pragma once

#include <iostream>

#include <vector>

namespace lab3 // https://youtu.be/IKJHMhtMnR0

{

using namespace std;

enum class direction { right\_direction = 0, down\_direction };

using matrix\_t = vector<vector<double>>;

struct SquareMatrix

{

private:

matrix\_t matrix;

public:

SquareMatrix(int size);

// методы доступа

int size() { return matrix.size(); }

double& get\_cell(int x, int y)

{

if ((x < matrix.size() && x >= 0) && (y < matrix.size() && y >= 0))

{

return matrix[y][x];

}

throw std::out\_of\_range("indexs\_out\_of\_range");

}

static void print\_matrix(SquareMatrix& matrix);

};

SquareMatrix fill\_matrix(SquareMatrix& matrix);

namespace math\_operation

{

SquareMatrix transpose(SquareMatrix& matrix);

SquareMatrix minor\_matrix(SquareMatrix matrix, int x\_rm, int y\_rm);

double determinant(SquareMatrix matrix);

SquareMatrix\* inverse\_matrix(SquareMatrix matrix);

SquareMatrix\* multiply\_matrix(SquareMatrix matrix1, SquareMatrix matrix2);

}

bool run(void);

void lab(void);

}

Файл «lab2.cpp»

#include "lab3.h"

using namespace lab3;

SquareMatrix::SquareMatrix(int size)

{

for (int i = 0; i < size; i++)

{

matrix.push\_back(vector<double>());

for (int k = 0; k < size; k++) matrix[i].push\_back(double(0));

}

}

void SquareMatrix::print\_matrix(SquareMatrix& matrix)

{

for (int i = 0; i < matrix.size(); i++)

{

for (int k = 0; k < matrix.size(); k++) std::cout << matrix.get\_cell(k, i) << "\t";

std::cout << endl << endl;

}

}

SquareMatrix math\_operation::transpose(SquareMatrix& matrix)

{

SquareMatrix result(matrix.size());

for (int y = 0; y < matrix.size(); y++)

{

for (int x = 0; x < matrix.size(); x++) result.get\_cell(x, y) = matrix.get\_cell(y, x);

}

return result;

}

SquareMatrix math\_operation::minor\_matrix(SquareMatrix matrix, int x\_rm, int y\_rm)

{

SquareMatrix result(matrix.size() - 1);

for (int y = 0, result\_y = 0; y < matrix.size(); y++)

{

if (y == y\_rm) continue;

for (int x = 0, result\_x = 0; x < matrix.size(); x++)

{

if (x == x\_rm) continue;

result.get\_cell(result\_x++, result\_y) = matrix.get\_cell(x, y);

}

result\_y++;

}

return result;

}

double math\_operation::determinant(SquareMatrix matrix)

{

// терминальное условие: если минор матрицы 2-ого порядка

if (matrix.size() <= 2)

{

return matrix.get\_cell(0, 0) \* matrix.get\_cell(1, 1)

- (matrix.get\_cell(1, 0) \* matrix.get\_cell(0, 1));

}

double result = 0;

for (int i = 0; i < matrix.size(); i++)

{

// находим минор

SquareMatrix new\_matrix = minor\_matrix(matrix, i, 0);

result += (i % 2 == 0 ? 1 : -1) \* matrix.get\_cell(i, 0) \* determinant(new\_matrix);

}

return result;

}

SquareMatrix\* math\_operation::inverse\_matrix(SquareMatrix matrix)

{

double det = determinant(matrix);

if (det == 0) return NULL;

SquareMatrix\* result = new SquareMatrix(matrix.size());

matrix = transpose(matrix);

for (int y = 0; y < result->size(); y++)

{

for (int x = 0; x < result->size(); x++)

{

result->get\_cell(x, y) = pow(-1, x + y) \* determinant(minor\_matrix(matrix, x, y)) \* (1. / det);

}

}

return result;

}

SquareMatrix\* math\_operation::multiply\_matrix(SquareMatrix matrix1, SquareMatrix matrix2)

{

if (matrix1.size() != matrix2.size()) return NULL;

SquareMatrix\* result = new SquareMatrix(matrix1.size());

// лямда-фукнция для вычисления значения клетки при перемножении столбцов и строк

// захватываем перемножаемые матрицы в анонимном методе

auto line\_operation = [&matrix1, &matrix2](int x, int y) -> double

{

double cell\_value = 0;

for (int i = 0; i < matrix1.size(); i++)

{

cell\_value += (matrix1.get\_cell(i, y) \* matrix2.get\_cell(x, i));

}

return cell\_value;

};

for (int y = 0; y < result->size(); y++)

{

for (int x = 0; x < result->size(); x++)

{

result->get\_cell(x, y) = line\_operation(x, y);

}

}

return result;

}

SquareMatrix lab3::fill\_matrix(SquareMatrix& matrix)

{

double counter = 1;

// установка значения для нулевой клетки (х = 0; у = 0)

matrix.get\_cell(0, 0) = counter++;

// установка начального навравления = направо

direction dir(direction::right\_direction);

// заполнение матрицы

for (int x = 0, y = 0; y != matrix.size() - 1 || x != matrix.size() - 1; )

{

// проверка: если направление = направо ; и справа свободно

if (x < matrix.size() - 1 && dir == direction::right\_direction)

{

// смещение курсора вправо и установка значения клетки

matrix.get\_cell(++x, y) = counter++;

// смена направления

dir = direction::down\_direction;

}

else if (y < matrix.size() - 1 && dir == direction::down\_direction)

{

matrix.get\_cell(x, ++y) = counter++;

dir = direction::right\_direction;

}

// если предыдущие условия не сработали то мы меняем направление движения курсора и повторяем (итерацию) еще раз

else

{

if (dir == direction::down\_direction) dir = direction::right\_direction;

else dir = direction::down\_direction;

continue;

}

// проверка если курсор находится на границе матрицы : слево или снизу

// если истина то это означает что движение по диагонали будет направлено по диагонали - вверх и вправо

// иначе - вниз и влево

if (x == 0 || y == matrix.size() - 1)

{

// перемещение курсора по диагонали пока он не упрется в стенку

// паралельно устанавливаются значения для клеток

while (x < matrix.size() - 1 && y > 0) matrix.get\_cell(++x, --y) = counter++;

}

else if (y == 0 || x == matrix.size() - 1)

{

while (y < matrix.size() - 1 && x > 0) matrix.get\_cell(--x, ++y) = counter++;

}

}

return matrix;

}

bool lab3::run(void)

{

int n = 0;

bool error\_state = true;

std::cout << fixed;

std::cout.precision(2);

try

{

while (true)

{

system("cls");

std::cout << "Введите размер матрицы (N x N) где 4 <= n <= 16: ";

cin >> n;

std::cout << endl;

if (n >= 4 && n <= 16) break;

else

{

std::cout << "Введены неверные значения" << endl;

system("pause");

}

}

SquareMatrix matrix(n), \* inverse, \* result;

matrix = fill\_matrix(matrix);

// определитель отличен от нуля, следовательно, матрица является невырожденной и для нее можно найти обратную матрицу.

if ((inverse = math\_operation::inverse\_matrix(matrix)) == NULL)

{

std::cout << "Исходная матрица является вырожденной - нельзя получить обратную матрицу" << endl;

throw std::exception("det = 0");

}

std::cout << "Сгенерированная матрица: " << endl << endl;

SquareMatrix::print\_matrix(matrix);

std::cout << endl << endl << "Обратная матрица: " << endl << endl;

SquareMatrix::print\_matrix(\*inverse);

std::cout << endl << endl;

if ((result = math\_operation::multiply\_matrix(matrix, \*inverse)) == NULL)

{

std::cout << "Матрицы нельзя перемножить" << endl;

throw std::exception("cannot multiply");

}

std::cout << endl << endl

<< "Полученная путем перемножения сгенерированной и обратной,\nЕдиничная матрица: "

<< endl << endl;

SquareMatrix::print\_matrix(\*result);

delete inverse, result;

}

catch (...)

{

error\_state = false;

}

return error\_state;

}

void lab3::lab(void)

{

bool error\_state = run();

}

Файл «main.cpp»

#include <iostream>

#include <locale>

#include "labs/lab3/lab3.h"

int main(void)

{

setlocale(LC\_ALL, "rus");

lab3::lab();

return 0;

}