ГУАП

КАФЕДРА №14

ОТЧЕТ

ЗАЩИЩЕН С ОЦЕНКОЙ

ПРЕПОДАВАТЕЛЬ

Должность, уч. степень, звание подпись, дата инициалы, фамилия

**ОТЧЕТ О ЛАБОРАТОРНОЙ РАБОТЕ №1**

по курсу: ТЕХНОЛОГИЯ ПРОГРАММИРОВАНИЯ

РАБОТУ ВЫПОЛНИЛ

СТУДЕНТ ГР. 1441 М.И. Лубинец

подпись, дата инициалы, фамилия

Санкт-Петербург

2015

1. Формализация задачи

Задача №1:

Для заданного целого N (0 <= N <= 40) создает квадратную матрицу

размера N на N такую, что каждый недиагональный элемент матрицы

представляет собой сумму индексов элемента (нумерация индексов начинается с

единицы), а каждый диагональный элемент - сумму элементов, находящихся с

ним в одной строке и одном столбце.

Отобразить матрицу на экране

Попернуть матрицу на 90 градусов по часовой стрелке и отобразить ее на экране

Задача №2

Реализовать класс, выполняющий условия задачи №1,

Добавить "безопасные" методы по работе с данными матрицы.

Добавить методы по установке и изменению размера матриц.

Добавить метод разворота матрицы на угол 90 \* n градусов, где n - целое.

Добавить метод Assign, который должен использоваться для копирования содержимого

одной матрицы в другую.

При реализации необходимо учитывать возможные ошибки при выделении памяти, некорректные

входные параметры и т.д. Программа должны вести себя адекватным образом и в случае

возникновения ошибок. Использование программы не должно приводить к утечке памяти.

Задача №3

Для класса матрицы из задачи №2, ереопределить бинарные математические операторы так, чтобы в арифметических операциях могли участвовать матрицы, натуральные числа.

Задача №5

Для класса матрицы, определить метод расчета определителя матрицы

2. Листинги

Задача 1.

Файл main.cpp

#include <iostream>

#include <stdint.h>

#include <assert.h>

#include <string>

/// Function computing elements in row and collumn

**inline** uint16\_t

row\_col\_sum(uint16\_t\*\* matrix, size\_t msize, uint16\_t num) {

assert(matrix != NULL && msize <= 40);

uint16\_t sum = 0;

/// Go through collumn

**for**(uint16\_t y = 0; y < msize; y++) {

sum += matrix[y][num];

}

/// Go through row

**for**(uint16\_t x = 0; x < msize; x++) {

sum += matrix[num][x];

}

**return** sum;

}

uint16\_t\*\*

init\_matrix(size\_t msize) {

assert(msize <= 40 && "0 <= size <= 40");

/// Creating array of pointers (collumns)

uint16\_t\*\* matrix = **new** uint16\_t\*[msize];

/// Initializing values in matrix

**for**(uint16\_t i = 0; i < msize; i++) {

matrix[i] = **new** uint16\_t[msize];

**for**(uint16\_t j = 0; j < msize; j++) {

**if**(i == j) {

matrix[i][j] = 0;

} **else** {

matrix[i][j] = (i+1) + (j+1);

}

}

}

/// Initializing main diagonal

**for**(uint16\_t i = 0; i < msize; i++) {

matrix[i][i] = row\_col\_sum(matrix, msize, i);

}

**return** matrix;

}

**void**

rotate\_90(uint16\_t\*\* matrix, size\_t msize) {

assert(matrix != NULL && msize <= 40);

uint16\_t temp;

**for**(uint16\_t i = 0; i < msize/2; i++) {

**for**(uint16\_t j = 0; j < msize/2; j++) {

temp = matrix[msize-1-j][i];

matrix[msize-1-j][i] = matrix[msize-1-i][msize-1-j];

matrix[msize-1-i][msize-1-j] = matrix[j] [msize-1-i];

matrix[j] [msize-1-i] = matrix[i] [j];

matrix[i] [j] = temp;

}

}

}

**void**

print\_matrix(uint16\_t\*\* matrix, size\_t msize) {

assert(matrix != NULL && msize <= 40);

std::string out;

**for**(uint16\_t i = 0; i < msize; i++) {

**for**(uint16\_t j = 0; j < msize; j++) {

out = std::to\_string(matrix[i][j]);

out.resize(4, ' ');

std::cout << out;

}

std::cout<< "\n\n";

}

}

**int** main() {

**const** size\_t msize = 40;

uint16\_t\*\* matrix = init\_matrix(msize);

print\_matrix(matrix, msize);

rotate\_90(matrix, msize);

print\_matrix(matrix, msize);

**return** 0;

}

Задача 2.

Файл matrix.hpp

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

**ModuleName:**

**Matrix**

**Abstract:**

**Matrix** **class,** **designed** **for** **Programming** **Technology** **class,** **extended** **version.**

**Author:**

**Lubinets** **Mike** **1441** **(Лубинец** **Михаил** **1441)**

**Date:**

**2016-03-20**

**\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/**

#ifndef PT\_L1\_T2\_MATRIX\_H

#define PT\_L1\_T2\_MATRIX\_H

#include <string>

#include <stdint.h>

#include <stddef.h>

**namespace** PT\_l1\_t2 {

**using** std::string;

**class** Matrix {

**public**:

Matrix();

**virtual** ~*Matrix*();

/\* @brief Create and initialize matrix with passed size

**\*** **@param** **size** **--** **desired** **matrix** **size** **\*/**

**void** Init(size\_t size);

**/\*** **@brief** **Rotate** **matrix** **by** **90** **degrees** **\*/**

**void** Rotate();

/\* @brief Output matrix to stdout \*/

**void** Print();

/\* Getters \*/

size\_t size() **const**;

**protected**:

**void** InitMemory();

**void** FreeMemory();

**int** ColRowSum(size\_t n);

size\_t \_size;

**int**\*\* \_matrix; ///< Array of pointers to \_memory (columns)

**int**\* \_memory; ///< Continuous chunk of memory (to prevent memory fragmentation)

};

}

#endif // PT\_L1\_T2\_MATRIX\_H

Файл matrix2.hpp

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

**ModuleName:**

**Matrix2**

**Abstract:**

**Matrix** **class,** **designed** **for** **Programming** **Technology** **class,** **extended** **version.**

**Author:**

**Lubinets** **Mike** **1441** **(Лубинец** **Михаил** **1441)**

**Date:**

**2016-03-20**

**Changes:**

**Addded** **safe** **operator[][]** **semantics**

**Resize** **method**

**Rotate** **by** **n\*90**

**Copy** **constructor** **and** **Assign** **operator**

**\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/**

#ifndef PT\_L1\_T2\_MATRIX2\_H

#define PT\_L1\_T2\_MATRIX2\_H

#include "matrix.hpp"

**namespace** PT\_l1\_t2 {

**class** Matrix2 : **public** Matrix {

**public**:

Matrix2();

Matrix2(**const** Matrix2& copy\_from);

Matrix2(**const** Matrix2\* copy\_from);

/\* @brief Copy matrix \*/

**void** Assign(**const** Matrix2& assign\_from);

**void** Assign(**const** Matrix2\* assign\_from);

/\* @brief Resize matrix

**\*** **@param** **new\_size** **--** **new** **matrix** **size** **\*/**

**void** Resize(size\_t new\_size);

/\* @brief Rotate matrix by n\*90 grad

**\*** **n** **can** **be** **positive** **or** **negative,** **but** **it** **only** **makes** **sence** **to** **pass** **1** **<=** **n** **<** **3,**

**\*** **as** **n** **>** **3** **will** **be** **translated** **to** **n** **%** **4,**

**\*** **-3** **=>** **1,** **-2** **=>** **2,** **-1** **=>** **3** **\*/**

**void** Rotate(**int** n);

/\* Proxy class for safe implementation of operator[][] semantics \*/

**struct** MatrixRow {

**explicit** MatrixRow(**int**\* row\_ptr, size\_t size);

**int**& **operator**[](size\_t ix);

**private**:

**int**\* \_ptr;

size\_t \_size;

};

/\* Safe access to matrix elements \*/

MatrixRow **operator**[](size\_t iy);

};

}

#endif // PT\_L1\_T2\_MATRIX2\_HPP

Файл matrix.cpp

#include "matrix.hpp"

#include <iostream>

#include <stdexcept>

**namespace** PT\_l1\_t2 {

Matrix::Matrix() :

\_size(0),

\_matrix(**nullptr**),

\_memory(**nullptr**) {}

Matrix::~*Matrix*() {

FreeMemory();

}

**void**

Matrix::Init(size\_t size) {

**if**(size > 40) {

**throw** std::logic\_error("ERROR: Size range must be: 0 <= size <= 40");

}

\_size = size;

InitMemory();

/// Do values initialization

**for**(uint i = 0; i < size; i++) {

**for**(uint j = 0; j < size; j++) {

**if**(i == j) {

\_matrix[i][j] = 0;

} **else** {

\_matrix[i][j] = (i+1) + (j+1);

}

}

}

/// Initializing main diagonal

**for**(uint i = 0; i < size; i++) {

\_matrix[i][i] = ColRowSum(i);

}

}

**void**

Matrix::Rotate() {

**if**(!\_matrix || !\_memory) {

**throw** std::logic\_error("ERROR: Matrix was not initialized, can't rotate");

}

**for**(uint y = 0; y < \_size/2; y++) {

size\_t s = \_size;

**for**(uint x = 0; x < s/2; x++) {

**int** temp = \_matrix[s-1-x][y];

\_matrix[s-1-x][y] = \_matrix[s-1-y][s-1-x];

\_matrix[s-1-y][s-1-x] = \_matrix[x] [s-1-y];

\_matrix[x] [s-1-y] = \_matrix[y] [x];

\_matrix[y] [x] = temp;

}

}

}

**void**

Matrix::Print() {

**if**(!\_matrix || !\_memory) {

**throw** std::logic\_error("ERROR: Matrix was not initialized, can't print");

}

std::string out;

**for**(uint16\_t i = 0; i < \_size; i++) {

**for**(uint16\_t j = 0; j < \_size; j++) {

/// Pretty print

out = std::to\_string(\_matrix[i][j]);

out.resize(4, ' ');

std::cout << out;

}

std::cout << "\n\n";

}

}

size\_t

Matrix::size() **const** {

**return** \_size;

}

**void**

Matrix::InitMemory() {

/// Checking if matrix was initialized previously to prevent memory leak

**if**(\_matrix || \_memory) {

FreeMemory();

}

/// Allocating memory for the matrix and collumns

\_memory = **new** int [\_size\*\_size];

\_matrix = **new** int\*[\_size];

/// Setting up collumn pointers

**for**(size\_t i = 0; i < \_size; i++) {

\_matrix[i] = \_memory + i\*\_size;

}

}

**void**

Matrix::FreeMemory() {

**if**(\_matrix != **nullptr**) {

std::clog << "TRACE: Freeing collumn pointers memory" << std::endl;

**delete**[] \_matrix;

}

**if**(\_memory != **nullptr**) {

std::clog << "TRACE: Freeing matrix memory" << std::endl;

**delete**[] \_memory;

}

}

**int**

Matrix::ColRowSum(size\_t n) {

uint16\_t sum = 0;

/// Go through collumn

**for**(uint y = 0; y < \_size; y++) {

sum += \_matrix[y][n];

}

/// Go through row

**for**(uint x = 0; x < \_size; x++) {

sum += \_matrix[n][x];

}

**return** sum;

}

}

Файл matrix2.cpp

#include "matrix2.hpp"

#include <stdexcept>

**namespace** PT\_l1\_t2 {

**using** std::to\_string;

Matrix2::Matrix2() : Matrix() {}

Matrix2::Matrix2(**const** Matrix2\* copy\_from) : Matrix2(\*copy\_from) {}

Matrix2::Matrix2(**const** Matrix2& copy\_from) : Matrix(copy\_from) {

Assign(copy\_from);

}

**void**

Matrix2::Assign(**const** Matrix2\* from) {

**return** Assign(\*from);

}

**void**

Matrix2::Assign(**const** Matrix2& from) {

\_size = from.size();

InitMemory();

size\_t mem\_size = \_size\*\_size;

**for**(size\_t i = 0; i < mem\_size; i++) {

\_memory[i] = from.\_memory[i];

}

}

**void**

Matrix2::Resize(size\_t new\_size) {

\_size = new\_size;

InitMemory();

}

**void**

Matrix2::Rotate(**int** n) {

**bool** neg = (n < 0) ? **true** : **false**;

n = abs(n) % 4;

**if**(n == 0 || n == 4) {

**return**;

}

n = (neg) ? 4 - n

: n;

**for**(**int** i = 0; i < n; i++) {

Matrix::Rotate();

}

}

Matrix2::MatrixRow::MatrixRow(**int**\* row\_ptr, size\_t size)

: \_ptr(row\_ptr), \_size(size) {}

**int**&

Matrix2::MatrixRow::**operator**[](size\_t ix) {

**if**(ix >= \_size) {

**throw** std::out\_of\_range (

"ERROR: Row index " + to\_string(ix) + " is out-of-range (0.." + to\_string(\_size-1) + ")"

);

}

**return** \_ptr[ix];

}

Matrix2::MatrixRow

Matrix2::**operator**[](size\_t iy) {

**if**(iy >= \_size) {

**throw** std::out\_of\_range (

"ERROR: Col index " + to\_string(iy) + " is out-of-range (0.." + to\_string(\_size-1) + ")"

);

}

**return** MatrixRow(\_matrix[iy], \_size);

}

}

Файл task2.cpp

#include <iostream>

#include "matrix.hpp"

#include "matrix2.hpp"

**using** **namespace** std;

**using** **namespace** PT\_l1\_t2;

**void** demo\_matrix() {

Matrix m1;

/// Trying to print and rotate non-initialized matrix

**try** { m1.Rotate(); } **catch**(std::exception& e) {

std::cerr << e.*what*();

}

**try** { m1.Print(); } **catch**(std::exception& e) {

std::cerr << e.*what*();

}

/// Trying to initialize woth wrong values

**try** { m1.Init(-1); } **catch**(std::exception& e) {

std::cerr << e.*what*();

}

**try** { m1.Init(41); } **catch**(std::exception& e) {

std::cerr << e.*what*();

}

/// Finished trolling matrix class, testing main functionality

m1.Init(10);

cout << "Matrix with size " << m1.size() << endl;

m1.Print();

cout << endl << endl

<< "Rotated by 90 deg matrix with size " << m1.size() << endl;

m1.Rotate();

m1.Print();

/// Reinitializing the matrix with size 15

m1.Init(15);

cout << "Matrix with size " << m1.size() << endl;

m1.Print();

cout << endl << endl

<< "Rotated by 90 deg matrix with size " << m1.size() << endl;

m1.Rotate();

m1.Print();

}

**void** demo\_matrix2() {

Matrix2 m1;

Matrix2 m2;

Matrix2 m3;

m1.Init(2);

m2.Init(4);

m3.Init(10);

cout << "Original matrix 2:\n";

m2.Print();

cout << "\nMatrix 2 rotated 2 times:\n";

m2.Rotate(2);

m2.Print();

cout << "\nOriginal matrix 3:\n";

m3.Print();

cout << "\nMatrix 3 rotated 400000001 times:\n";

m3.Rotate(400000001);

m3.Print();

cout << "\nOriginal matrix 1:\n";

m1.Print();

cout << "\nSet 0:1 and 1:1 in matrix 1 to 0:\n";

m1[0][1] = 0;

m1[1][1] = 0;

m1.Print();

cout << "\nTrying to set 10:1 of matrix 1:\n";

**try** {

m1[10][1] = 0;

} **catch**(std::out\_of\_range& e) {

std::cerr << e.*what*();

}

cout << "\nTrying to set 1:10 of matrix 1:\n";

**try** {

m1[1][10] = 0;

} **catch**(std::out\_of\_range& e) {

std::cerr << e.what();

}

cout << "\nCopy matrix 1 to matrix 3:\n";

m3.Assign(m1);

m3.Print();

}

**int** main() {

demo\_matrix();

demo\_matrix2();

size\_t matrix\_size;

cout << "Enter matrix size: ";

cin >> matrix\_size;

Matrix m1;

**try** {

m1.Init(matrix\_size);

} **catch** (std::exception& e) {

std::cout << e.*what*();

**return** 1;

}

cout << "Matrix with size " << m1.size() << endl;

m1.Print();

cout << endl << endl

<< "Rotated by 90 deg matrix with size " << m1.size() << endl;

m1.Rotate();

m1.Print();

}

Задача 3

Файл martix3.hpp

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

**ModuleName:**

**Matrix**

**Abstract:**

**Matrix** **class,** **designed** **for** **Programming** **Technology** **class,** **double** **extended** **version.**

**Author:**

**Lubinets** **Mike** **1441** **(Лубинец** **Михаил** **1441)**

**Date:**

**2016-03-20**

**Changes**

**Arithmetic** **operators**

**\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/**

#ifndef MATRIX3\_H

#define MATRIX3\_H

#include "matrix2.hpp"

**namespace** PT\_l1\_t3 {

**class** Matrix3 : **public** PT\_l1\_t2::Matrix2 {

**public**:

**using** Matrix2::Matrix2;

Matrix3& **operator**= (Matrix3& b);

Matrix3& **operator**= (**int** b);

/\* Operations with matrices \*/

**friend** Matrix3 **operator**+ (**const** Matrix3& a, **const** Matrix3& b);

**friend** Matrix3 **operator**- (**const** Matrix3& a, **const** Matrix3& b);

**friend** Matrix3 **operator**\* (**const** Matrix3& a, **const** Matrix3& b);

/\* Operations with natural numbers \*/

**friend** Matrix3 **operator**\* (**const** Matrix3& a, **int** b);

};

}

#endif // MATRIX3\_H

Файл matrix3.cpp

#include "matrix3.hpp"

#include <stdexcept>

**namespace** PT\_l1\_t3 {

Matrix3&

Matrix3::**operator**= (Matrix3& b) {

Assign(b);

**return** \***this**;

}

Matrix3 **operator**+ (**const** Matrix3& a, **const** Matrix3& b) {

**if**(a.size() != b.size()) {

**throw** std::logic\_error("a.size() != b.size()");

}

size\_t size = a.size();

size\_t msize = size\*size;

Matrix3 c;

c.Resize(size);

**for**(uint i = 0; i < msize; i++) {

c.\_memory[i] = a.\_memory[i] + b.\_memory[i];

}

**return** c;

}

Matrix3 **operator**- (**const** Matrix3& a, **const** Matrix3& b) {

**if**(a.size() != b.size()) {

**throw** std::logic\_error("a.size() != b.size()");

}

size\_t size = a.size();

size\_t msize = size\*size;

Matrix3 c;

c.Resize(size);

**for**(uint i = 0; i < msize; i++) {

c.\_memory[i] = a.\_memory[i] - b.\_memory[i];

}

**return** c;

}

Matrix3 **operator**\* (**const** Matrix3& a, **const** Matrix3& b) {

**if**(a.size() != b.size()) {

**throw** std::logic\_error("a.size() != b.size()");

}

size\_t size = a.size();

Matrix3 c;

c.Resize(size);

**for**(uint i = 0; i < size; i++) {

**for**(uint j = 0; j < size; j++) {

**int** c\_ij = 0;

**for**(uint k = 0; k < size; k++) {

c\_ij += a.\_matrix[i][k] + a.\_matrix[k][j];

}

c[i][j] = c\_ij;

}

}

**return** c;

}

Matrix3 **operator**\* (**const** Matrix3& a, **int** b) {

size\_t size = a.size();

size\_t msize = size\*size;

Matrix3 c;

c.Resize(size);

**for**(uint i = 0; i < msize; i++) {

c.\_memory[i] = a.\_memory[i] \* b;

}

**return** c;

}

}

Файл task3.cpp

#include <iostream>

#include "matrix3.hpp"

**using** **namespace** std;

**using** **namespace** PT\_l1\_t3;

**int** main() {

Matrix3 m1;

Matrix3 m2;

m1.Init(5);

m2.Init(5);

cout << "M1:\n";

m1.Print();

cout << "M2:\n";

m2.Print();

cout << "M1+M2=\n";

(m1+m2).Print();

cout << "M1-M2=\n";

(m1-m2).Print();

cout << "M1\*M2=\n";

(m1\*m2).Print();

cout << "M1\*10=\n";

(m1\*10).Print();

cout << "M3 = (M1 + M2)\n";

Matrix3 m3 = (m1 + m2);

m3.Print();

**return** 0;

}

Задание 5.1

Файл matrix4.hpp

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

**ModuleName:**

**Matrix**

**Abstract:**

**Matrix** **class,** **designed** **for** **Programming** **Technology** **class,** **triple** **extended** **version.**

**Author:**

**Lubinets** **Mike** **1441** **(Лубинец** **Михаил** **1441)**

**Date:**

**2016-03-21**

**Changes**

**Determinant** **calculation**

**\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/**

#ifndef MATRIX4\_H

#define MATRIX4\_H

#include "matrix3.hpp"

**namespace** PT\_l1\_t5 {

**class** Matrix4 : **public** PT\_l1\_t3::Matrix3 {

**public**:

**using** Matrix3::Matrix3;

**int** Determinant();

**private**:

**static** **int** Determinant(**int**\*\* matrix, size\_t size);

};

}

#endif // MATRIX4\_H

Файл matrix4.cpp

#include "matrix4.hpp"

#include <stdexcept>

#include <cmath>

**namespace** PT\_l1\_t5 {

**int**

Matrix4::Determinant() {

**if**(!\_memory || !\_matrix) {

**throw** std::logic\_error("Matrix must be initialized first");

}

**if**(\_size < 1) {

**throw** std::logic\_error("Matrix size mast be >= 1 to calculate discriminant");

}

**return** Determinant(\_matrix, \_size);

}

**int**

Matrix4::Determinant(**int**\*\* matrix, size\_t size) {

**if**(size == 1) {

**return** matrix[0][0];

}

**if**(size == 2) {

**return** matrix[0][0] \* matrix[1][1] - matrix[1][0] \* matrix[0][1];

}

**int** d = 0, k = 1;

#ifndef \_\_GNUG\_\_

**int**\*\* p = **new** **int**\*[size-1];

#**else**

**int**\* p[size-1];

#endif

uint j;

**for**(uint i = 0; i < size; i++) {

**for**(j = 0; j < size-1; j++) {

**if**(j < i) {

p[j] = matrix[j];

} **else** {

p[j] = matrix[j+1];

}

}

k = ((i+j) % 2) ? -1 : 1;

d += k \* Determinant(p, size-1) \* matrix[i][size-1];

}

#ifndef \_\_GNUG\_\_

**delete**[] p;

#endif

**return** d;

}

}

Файл task5.1.cpp

#include <iostream>

#include "matrix4.hpp"

**using** **namespace** std;

**using** **namespace** PT\_l1\_t5;

**int** main() {

Matrix4 m;

m.Init(5);

m.Print();

cout << "Det: " << m.Determinant() << endl;

}

3. Примеры

Задача 1:

Матрица 5х5 Повернутая матрица

|  |  |
| --- | --- |
| 36 3 4 5 6  3 42 5 6 7  4 5 48 7 8  5 6 7 54 9  6 7 8 9 60 | 6 5 4 3 36  7 6 5 42 3  4 5 48 7 8  9 54 7 6 5  60 9 8 7 6 |

Задача 2:

Ввод данных: 2

|  |  |
| --- | --- |
| 6 3  3 6 | 3 6  6 3 |

Задача 3:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| M1 | M2 | M1+M2 | M1-M2 | M1\*M2 | M1\*10 |
| 14 3 4  3 16 5  4 5 18 | 14 3 4  3 16 5  4 5 18 | 28 6 8  6 32 10  8 10 36 | 0 0 0  0 0 0  0 0 0 | 42 45 48  45 48 51  48 51 54 | 140 30 40  30 160 50  40 50 180 |

Задача 5.1:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 1x1 | 2x2 | 3x3 | 4x4 | 5x5 |
| 0 | 6 3  3 6 | 14 3 4  3 16 5  4 5 18 | 24 3 4 5  3 28 5 6  4 5 32 7  5 6 7 36 | 36 3 4 5 6  3 42 5 6 7  4 5 48 7 8  5 6 7 54 9  6 7 8 9 60 |
| Det = 0 | Det = 27 | Det = 3384 | Det = 669456 | Det = 206132400 |