**Московский государственный технический**

**университет им. Н.Э. Баумана**

Факультет «Информатика и системы управления»

Кафедра ИУ5 «Системы обработки информации и управления»

Курс «Основы информатики»

Отчет по лабораторной работе №10

«Вычисление обратной матрицы методом Гаусса-Жордана»

|  |  |  |
| --- | --- | --- |
| Выполнил: |  | Проверил: |
| студент группы ИУ5-11 |  | преподаватель каф. ИУ5 |
| Михалёв Ярослав |  | Козлов А.Д. |
| Подпись и дата: |  | Подпись и дата: |
|  |  |  |

Москва, 2021 г.

Постановка задачи

**Цели работы.**

- разработка программы, реализующей алгоритм вычисления обратной матрицы методом Гаусса-Жордана;

- использование разработанных функций для решения систем линейных уравнений.

**Задание.**

Создать функцию для вычисления обратной матрицы по методу Гаусса-Жордана. Размер матрицы передавать в функцию в качестве параметра. Для упрощения алгоритма следует присоединить единичную матрицу справа к исходной и выполнять все преобразования над объединенной матрицей размером N\*2N. Обратная матрица получится на месте единичной в столбцах N…2N, а на месте исходной матрицы в столбцах 0…(N-1) должна получиться единичная матрица.

Включить в алгоритм проверку на существование обратной матрицы. Для этого в прямом ходе перед делением выполнить проверку на ноль элементов главной диагонали исходной матрицы. Если элемент равен 0, то нужно поменять местами текущую строку с одной из нижележащих строк, в которой элемент в соответствующем столбце не равен 0.

Если таких строк нет, то выдать сообщение: «Обратная матрица не существует».

Применить функцию для решения системы линейных алгебраических уравнений.

Разработка алгоритма

**Описание входных, выходных и вспомогательных данных:**

Входные данные:

* int N – размер матрицы
* double\*\* A - матрица

Выходные данные:

* double det – определитель матрица A
* double\*\* A\_reverse – обратная матрица по отношению к A

**Список функций:**

double\*\* CreateMatrix(int N)

Возвращает единичную матрицу размером N - 1

Входные данные:

* int N – размер матрицы

Выходные данные:

* double\*\* A – единичная матрица

double\*\* CreateZeroMatrix(int N)

Возвращает нулевую матрицу размером N

Входные данные:

* int N – размер матрицы

Выходные данные:

* double\*\* A – нулевая матрица

double\*\* ReadMatrix(const char\* filename, int N)

Считывает матрицу из файла

Входные данные:

* const char\* filename – имя текстового файла
* int N – размер матрицы

Выходные данные:

* double\*\* A – матрица

double\*\* GenerateMatrix(int N)

Возвращает матрицу размером N из случайных элементов

Входные данные:

* int N – размер матрицы

Выходные данные:

* double\*\* A – матрица

double\*\* CloneMatrix(double\*\* A, int N)

Клонирует матрицу A размером N

Входные данные:

* double\*\* A – матрица, которую нужно скопировать
* int N – размер матрицы A

Выходные данные:

* double\*\* B – скопированная матрица

void PrintMatrix(double\*\* A, int N)

Выводит матрицу на экран

Входные данные:

* double\*\* A – матрица
* int N – размер N

void Clear(double\*\* A, int N)

Очищает из памяти двумерный динамический массив A

Входные данные:

* double\*\* A – динамический массив
* int N – размер динамического массива А

double\*\* Minor(double\*\* A, double\*\* p, int i, int j, int N)

Возвращает матрицу A без i-й строки и j-го столбца (Минор Aij)

Входные данные:

* double\*\* A – матрица A
* int N – размер матрицы A
* int i – число
* int j – число

Выходные данные:

* double\*\* B – матрица A без i-й строки и j-го столбца (Минор Aij)

double Determinant(double\*\* A, int N)

Возвращает определитель матрицы A размером N

Входные данные:

* double\*\* A – матрица A
* int N – размер матрицы A

Выходные данные:

* int d – определитель матрицы A

void Transposition(double\*\* A, int N)

Транспонирует матрицу A размером N

Входные данные:

* double\*\* A – матрица A
* int N – размер матрицы A

double\*\* Multiple(double\*\* A, double\*\* B, int N)

Возвращает матрицу, которая является произведением матриц A и B

Входные данные:

* double\*\* A – матрица A
* double\*\* B – матрица B
* int N – размер матриц A и B

double\*\* Inverse(double\*\* A, int N)

Возвращает обратную матрицу A

Входные данные:

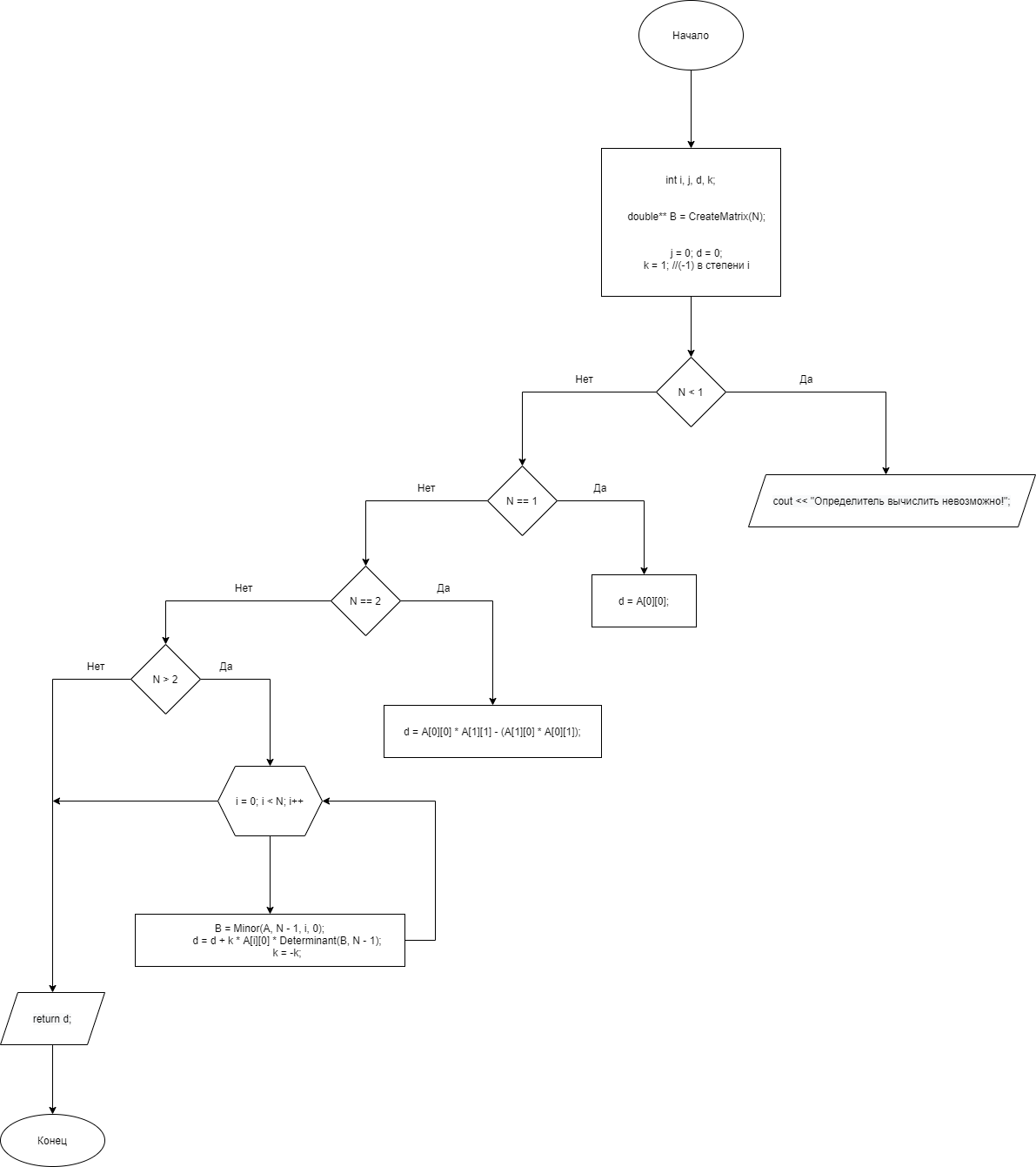
* double\*\* A – матрица A
* int N – размер матрицы A

Выходные данные:

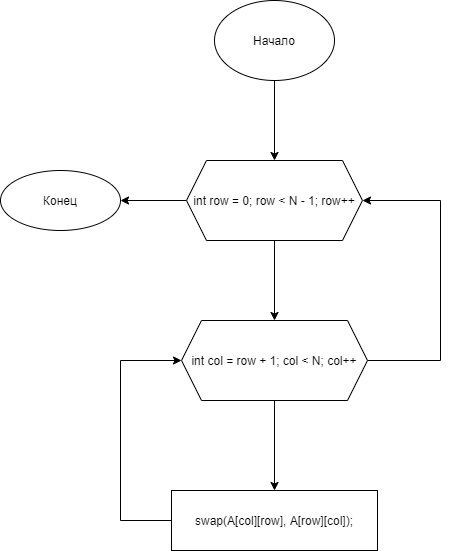
* double\*\* A\_reverse – обратная матрица A

Схема алгоритма

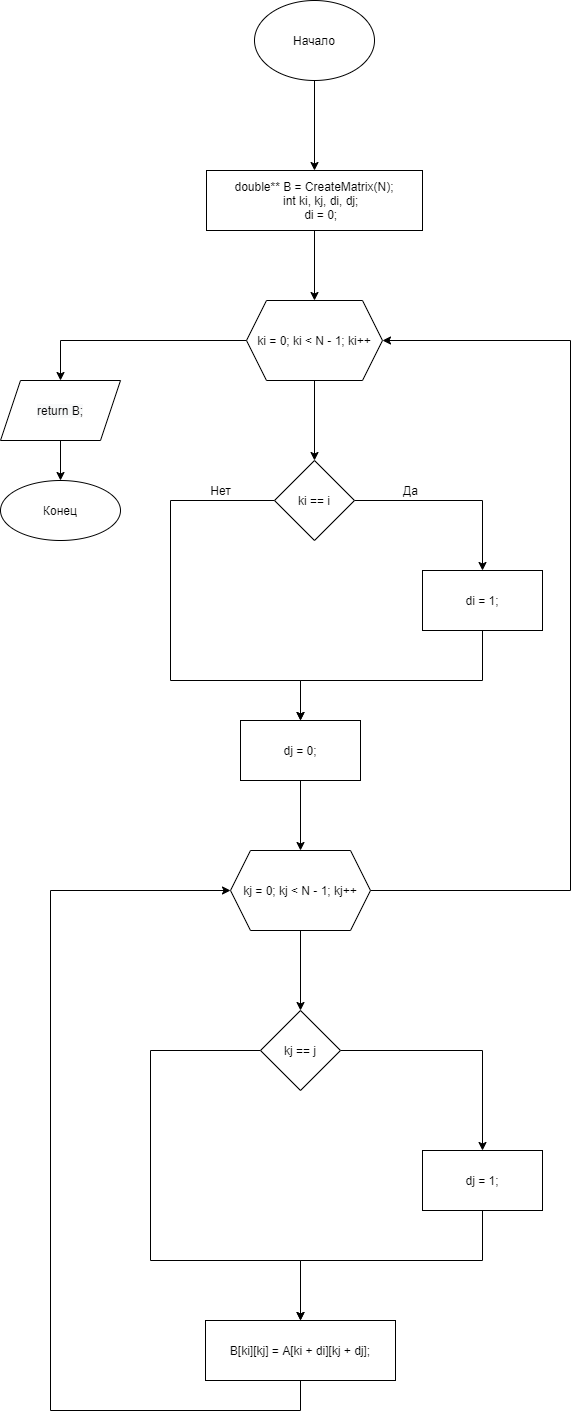
**Determinant**



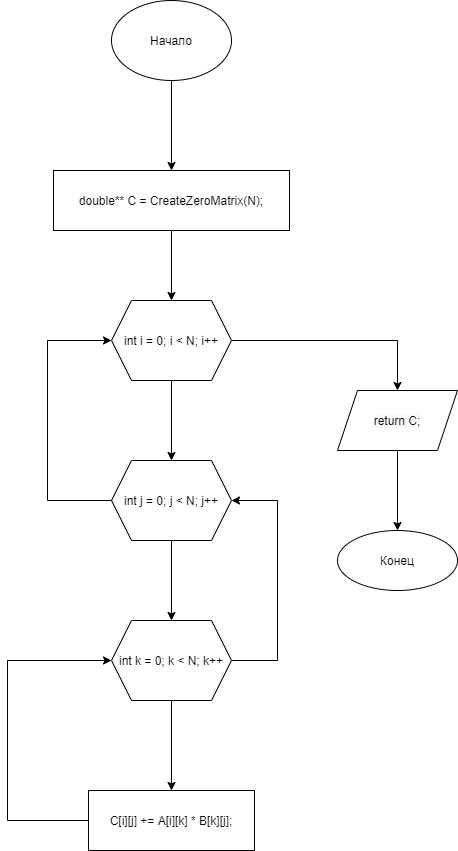
**Transposition**



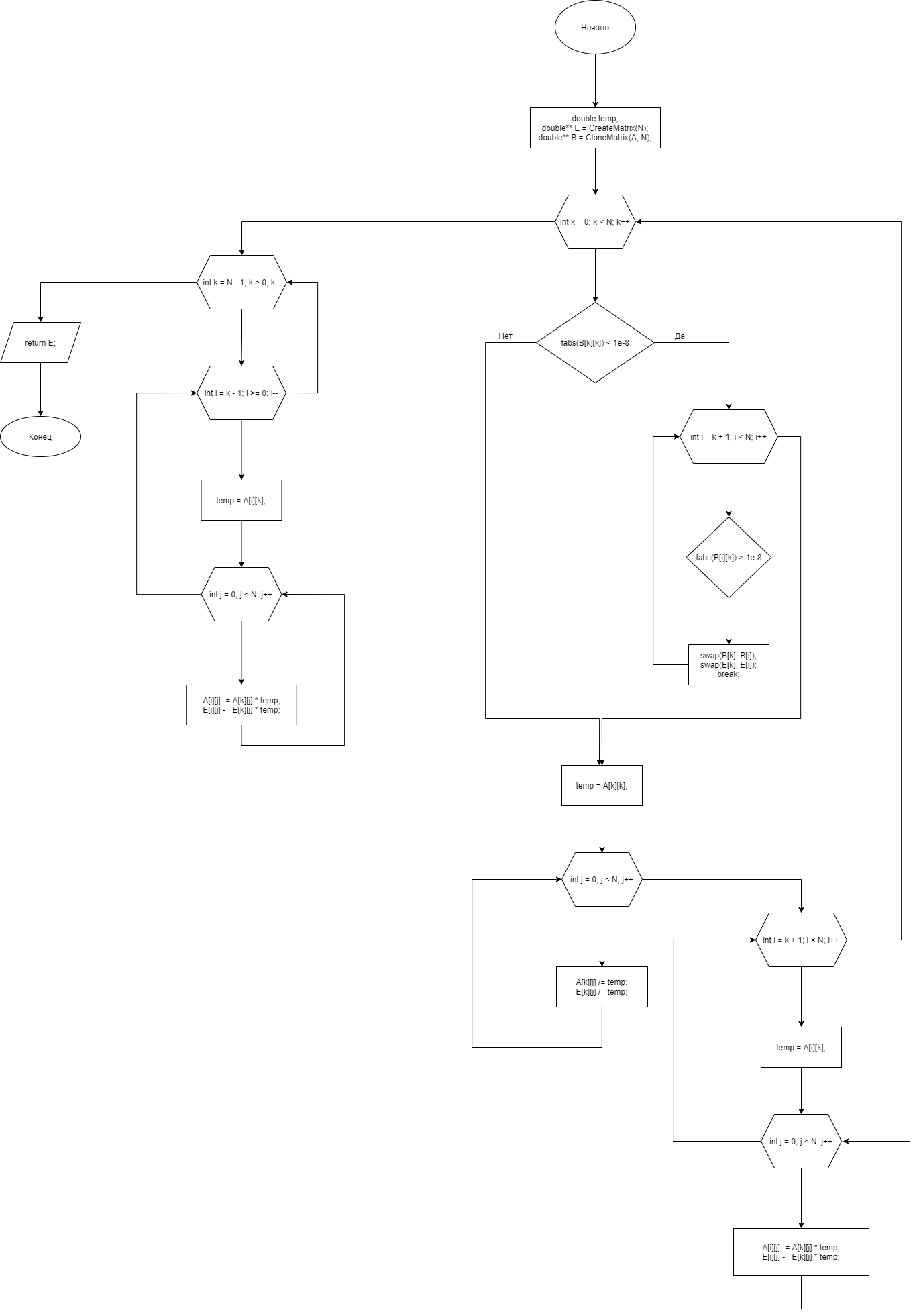
**Minor**



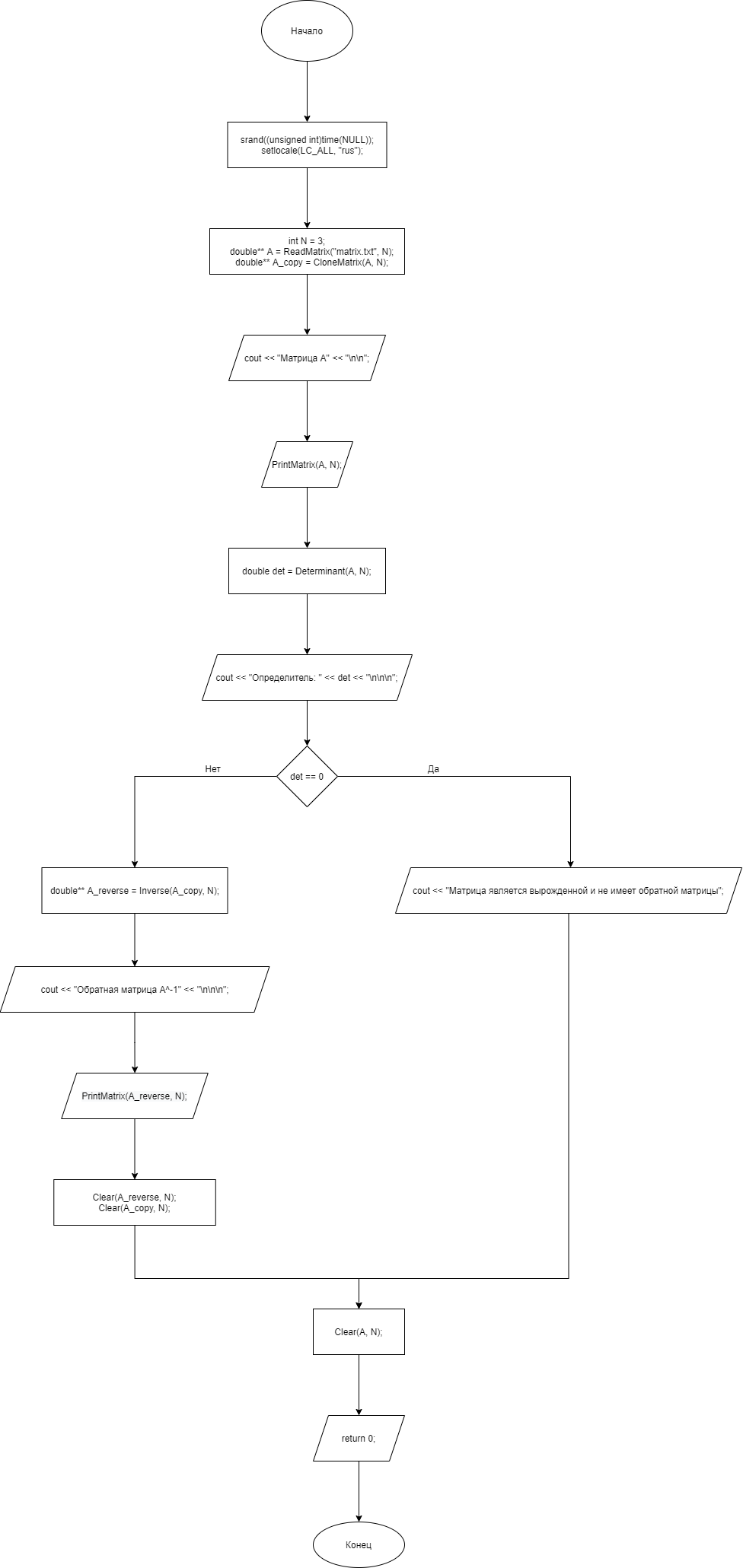
**Multiple**



**Inverse**

****

**Main**

****

Текст программы

Utils.cpp

#include <iostream>

#include <iomanip>

#include <fstream>

#include "Header.h"

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

// Создаёт единичную матрицу

double**\*\*** CreateMatrix**(**int N**)** **{**

double**\*\*** E **=** **new** double**\*** **[**N **+** 1**];**

**for** **(**int i **=** 0**;** i **<** N**;** i**++)**

**{**

E**[**i**]** **=** **new** double**[**N **+** 1**];**

**for** **(**int j **=** 0**;** j **<** N**;** j**++)**

E**[**i**][**j**]** **=** 0**;**

E**[**i**][**i**]** **=** 1**;**

**}**

**return** E**;**

**}**

// Создаёт нулевую матрицу

double**\*\*** CreateZeroMatrix**(**int N**)** **{**

double**\*\*** E **=** **new** double**\*** **[**N **+** 1**];**

**for** **(**int i **=** 0**;** i **<** N**;** i**++)**

**{**

E**[**i**]** **=** **new** double**[**N **+** 1**];**

**for** **(**int j **=** 0**;** j **<** N**;** j**++)**

E**[**i**][**j**]** **=** 0**;**

**}**

**return** E**;**

**}**

// Считывает матрицу из файла

double**\*\*** ReadMatrix**(**const char**\*** filename**,** int N**)**

**{**

double**\*\*** A **=** CreateMatrix**(**N**);**

ifstream file**;**

file**.**open**(**filename**);**

**for** **(**int i **=** 0**;** i **<** N**;** i**++)**

**for** **(**int j **=** 0**;** j **<** N**;** j**++)**

file **>>** A**[**i**][**j**];**

file**.**close**();**

**return** A**;**

**}**

// Создаёт матрицу из случайных элементов

double**\*\*** GenerateMatrix**(**int N**)** **{**

double**\*\*** A **=** **new** double**\*** **[**N **+** 1**];**

**for** **(**int i **=** 0**;** i **<** N**;** i**++)**

**{**

A**[**i**]** **=** **new** double**[**N **+** 1**];**

**for** **(**int j **=** 0**;** j **<** N**;** j**++)**

A**[**i**][**j**]** **=** rand**()** **%** 10**;**

**}**

**return** A**;**

**}**

// Вовращает копию матрицы

double**\*\*** CloneMatrix**(**double**\*\*** A**,** int N**)**

**{**

double**\*\*** B **=** **new** double**\*** **[**N**];**

**for** **(**int row **=** 0**;** row **<** N**;** row**++)**

**{**

B**[**row**]** **=** **new** double**[**N**];**

**for** **(**int col **=** 0**;** col **<** N**;** col**++)**

B**[**row**][**col**]** **=** A**[**row**][**col**];**

**}**

**return** B**;**

**}**

// Печать матрицы на экран

void PrintMatrix**(**double**\*\*** A**,** int N**)**

**{**

**for** **(**int i **=** 0**;** i **<** N**;** i**++)**

**{**

**for** **(**int j **=** 0**;** j **<** N**;** j**++)**

cout **<<** setprecision**(**2**)** **<<** left **<<** setw**(**16**)** **<<** A**[**i**][**j**];**

cout **<<** endl**;**

**}**

cout **<<** "\n\n\n"**;**

**}**

// Очищаем память

void Clear**(**double**\*\*** A**,** int N**)**

**{**

**for** **(**int i **=** 0**;** i **<** N**;** i**++)**

**delete[]** A**[**i**];**

**delete[]** A**;**

**}**

Main.cpp

#include <iostream>

#include <iomanip>

#include <fstream>

#include "Header.h"

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

// Возвращает матрицу A без i-й строки и j-го столбца

double**\*\*** Minor**(**double**\*\*** A**,** int N**,** int i**,** int j**)** **{**

double**\*\*** B **=** CreateMatrix**(**N**);**

int ki**,** kj**,** di**,** dj**;**

di **=** 0**;**

**for** **(**ki **=** 0**;** ki **<** N **-** 1**;** ki**++)** **{** // проверка индекса строки

**if** **(**ki **==** i**)** di **=** 1**;**

dj **=** 0**;**

**for** **(**kj **=** 0**;** kj **<** N **-** 1**;** kj**++)** **{** // проверка индекса столбца

**if** **(**kj **==** j**)** dj **=** 1**;**

B**[**ki**][**kj**]** **=** A**[**ki **+** di**][**kj **+** dj**];**

**}**

**}**

**return** B**;**

**}**

// Рекурсивное вычисление определителя

double Determinant**(**double**\*\*** A**,** int N**)** **{**

int i**,** j**,** d**,** k**;**

double**\*\*** B **=** CreateMatrix**(**N**);**

j **=** 0**;** d **=** 0**;**

k **=** 1**;** //(-1) в степени i

**if** **(**N **<** 1**)** cout **<<** "Определитель вычислить невозможно!"**;**

**else** **if** **(**N **==** 1**)**

d **=** A**[**0**][**0**];**

**else** **if** **(**N **==** 2**)**

d **=** A**[**0**][**0**]** **\*** A**[**1**][**1**]** **-** **(**A**[**1**][**0**]** **\*** A**[**0**][**1**]);**

**else** **if** **(**N **>** 2**)** **{**

**for** **(**i **=** 0**;** i **<** N**;** i**++)** **{**

B **=** Minor**(**A**,** N**,** i**,** 0**);**

d **=** d **+** k **\*** A**[**i**][**0**]** **\*** Determinant**(**B**,** N **-** 1**);**

k **=** **-**k**;**

**}**

**}**

**return** d**;**

**}**

// Транспонирование матрицы

void Transposition**(**double**\*\*** A**,** int N**)**

**{**

**for** **(**int row **=** 0**;** row **<** N **-** 1**;** row**++)**

**for** **(**int col **=** row **+** 1**;** col **<** N**;** col**++)**

swap**(**A**[**col**][**row**],** A**[**row**][**col**]);**

**}**

// Перемножает две матрицы

double**\*\*** Multiple**(**double**\*\*** A**,** double**\*\*** B**,** int N**)**

**{**

double**\*\*** C **=** CreateZeroMatrix**(**N**);**

**for** **(**int i **=** 0**;** i **<** N**;** i**++)**

**for** **(**int j **=** 0**;** j **<** N**;** j**++)**

**for** **(**int k **=** 0**;** k **<** N**;** k**++)**

C**[**i**][**j**]** **+=** A**[**i**][**k**]** **\*** B**[**k**][**j**];**

**return** C**;**

**}**

// Возвращает обратную матрицу

double**\*\*** Inverse**(**double**\*\*** A**,** int N**)**

**{**

// Находим обратную матрицу методом алгебраических дополнений

double temp**;**

double**\*\*** E **=** CreateMatrix**(**N**);** // E - единичная матрица

double**\*\*** B **=** CloneMatrix**(**A**,** N**);**

/\*-------------- Прямой ход ---------------------------\*/

**for** **(**int k **=** 0**;** k **<** N**;** k**++)**

**{**

/\* Если элемент на главной диагонали в исходной строке = 0, то ищем строку, где

элемент того же столбца не нулевой, и меняем строки местами \*/

**if** **(**fabs**(**B**[**k**][**k**])** **<** 1e-8**)**

**{**

**for** **(**int i **=** k **+** 1**;** i **<** N**;** i**++)**

**{**

**if** **(**fabs**(**B**[**i**][**k**])** **>** 1e-8**)**

**{**

swap**(**B**[**k**],** B**[**i**]);**

swap**(**E**[**k**],** E**[**i**]);**

**break;**

**}**

**}**

**}**

temp **=** B**[**k**][**k**];**

**for** **(**int j **=** 0**;** j **<** N**;** j**++)**

**{**

B**[**k**][**j**]** **/=** temp**;**

E**[**k**][**j**]** **/=** temp**;**

**}**

**for** **(**int i **=** k **+** 1**;** i **<** N**;** i**++)**

**{**

temp **=** B**[**i**][**k**];**

**for** **(**int j **=** 0**;** j **<** N**;** j**++)**

**{**

B**[**i**][**j**]** **-=** B**[**k**][**j**]** **\*** temp**;**

E**[**i**][**j**]** **-=** E**[**k**][**j**]** **\*** temp**;**

**}**

**}**

**}**

/\*-----------------------------------------------------\*/

/\*

Привели матрицу A к ступенчатому виду

1 .........

0 1 .......

0 0 1 .....

0 0 0 1 ...

0 0 0 0 1 .

0 0 0 0 0 1

\*/

/\*-------------- Обратный ход -------------------------\*/

**for** **(**int k **=** N **-** 1**;** k **>** 0**;** k**--)**

**{**

**for** **(**int i **=** k **-** 1**;** i **>=** 0**;** i**--)**

**{**

temp **=** B**[**i**][**k**];**

**for** **(**int j **=** 0**;** j **<** N**;** j**++)**

**{**

B**[**i**][**j**]** **-=** B**[**k**][**j**]** **\*** temp**;**

E**[**i**][**j**]** **-=** E**[**k**][**j**]** **\*** temp**;**

**}**

**}**

**}**

/\*-----------------------------------------------------\*/

/\*

Привели матрицу A к единичному виду

1 0 0 0 0 0

0 1 0 0 0 0

0 0 1 0 0 0

0 0 0 1 0 0

0 0 0 0 1 0

0 0 0 0 0 1

E - обратная матрица по отношению к A

\*/

**return** E**;**

**}**

int main**()**

**{**

srand**((**unsigned int**)**time**(NULL));**

setlocale**(**LC\_ALL**,** "rus"**);**

int N **=** 4**;**

double**\*\*** A **=** ReadMatrix**(**"matrix.txt"**,** N**);**

cout **<<** "Матрица A" **<<** "\n\n"**;**

PrintMatrix**(**A**,** N**);**

double det **=** Determinant**(**A**,** N**);**

cout **<<** "Определитель: " **<<** det **<<** "\n\n\n"**;**

**if** **(**det **==** 0**)**

**{**

cout **<<** "Матрица является вырожденной и не имеет обратной матрицы"**;**

**}**

**else**

**{**

double**\*\*** A\_reverse **=** Inverse**(**A**,** N**);**

cout **<<** "Обратная матрица A^-1" **<<** "\n\n"**;**

PrintMatrix**(**A\_reverse**,** N**);**

// Проверка

cout **<<** "Проверка A \* A^-1" **<<** "\n\n"**;**

PrintMatrix**(**Multiple**(**A**,** A\_reverse**,** N**),** N**);**

Clear**(**A\_reverse**,** N**);**

**}**

Clear**(**A**,** N**);**

**return** 0**;**

**}**

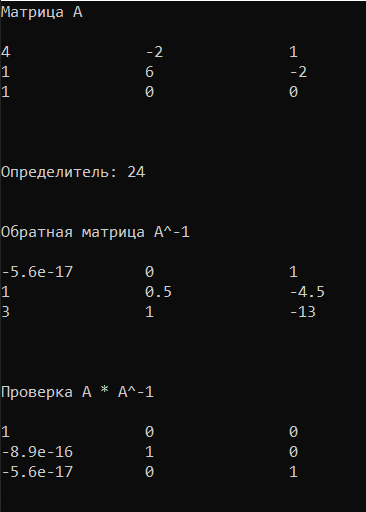
Анализ результатов

Для матрицы

4 -2 1

1 6 -2

1 0 0



Для матрицы

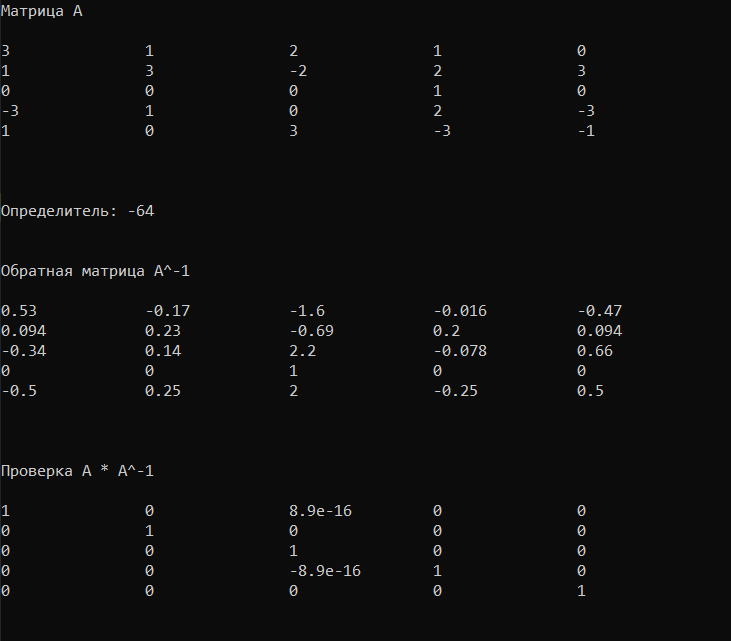
3 1 2 1 0

1 3 -2 2 3

0 0 0 1 0

-3 1 0 2 -3

1 0 3 -3 -1



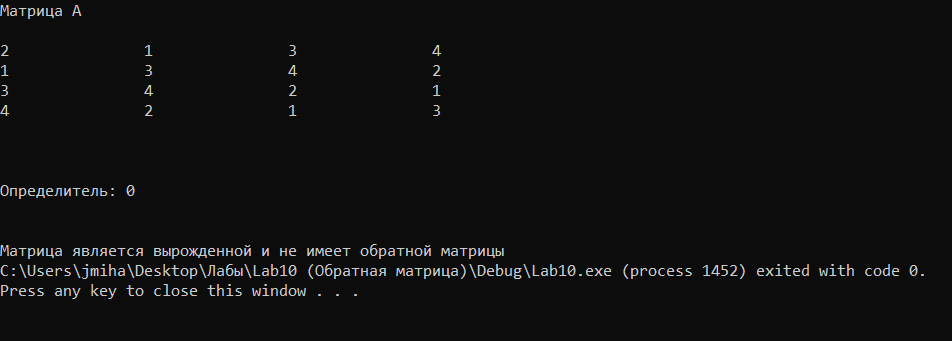
Для матрицы

2 1 3 4

1 3 4 2

3 4 2 1

4 2 1 3



Вывод

Я научился

* Работать с двумерными динамическими массивами
* Обрабатывать матрицы
* Использовать текстовые файлы для хранения данных
* Находить определитель матрицы
* Перемножать матрицы
* Находить обратную матрицу