МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное автономное образовательное учреждение

Высшего образования

**«Нижегородский государственный университет им. Н.И. Лобачевского»**

**Национальный исследовательский университет**

**Институт информационных технологий, математики и механики**

**Кафедра математического обеспечения и суперкомпьютерных технологий**

**Отчет по учебной практике**

**«Структура хранения матриц специального вида»**

**Выполнил:** студент группы 381706-1

Максимова Ирина Игоревна

\_\_\_\_\_\_\_\_\_\_\_ Подпись

**Научный руководитель:**

ассистент каф. МОСТ ИИТММ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Лебедев И.Г

Нижний Новгород

2018.

Содержание

[1. Введение 3](#_Toc531274519)

[2. Постановка задачи 4](#_Toc531274520)

[3. Руководство пользователя 5](#_Toc531274521)

[4. Руководство программиста 6](#_Toc531274522)

[4.1. Описание структуры программы 6](#_Toc531274523)

[4.2. Описание структур данных 7](#_Toc531274524)

[4.3. Описание алгоритмов 9](#_Toc531274525)

[5. Заключение 11](#_Toc531274526)

[6. Литература 12](#_Toc531274527)

# Введение

**Матрица –** математический объект, записываемый в виде прямоугольной таблицы элементов кольца или поля (например, целых, действительных или комплексных чисел), которая представляет собой совокупность строк и столбцов, на пересечении которых находятся её элементы.

Помимо прямоугольных матриц, для которых наиболее часто используемой представляется программная реализация в виде двумерного массива, в математических приложениях выделяются различные матрицы специальных видов (треугольные, диагональные и т.д.). Для таких матриц предпочтительно создание собственных способов хранения и обработки, учитывающих специфику их структуры. И с учетом этого они могут оказать более эффективными.

Верхнетреугольные матрицы, которые представляют собой квадратные матрицы, элементы которой ниже главной диагонали равны нулю

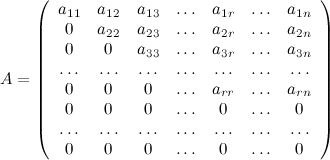


Рисунок 1. Верхнетреугольная матрица

**Цель данной лабораторной задачи** – разработка структуры хранения верхнетреугольных матриц.

# Постановка задачи

Выполнение работы предполагает решение следующих задач:

1. Разработка и реализация вспомогательного класса для создания верхнетреугольных матриц TVector.
2. Разработка и реализация класса матриц TMatrix
3. Создание класса MyException для обработки исключений, которые могут возникнуть при выполнении различных операций.
4. Программа, демонстрирующая работу классов TMatrix и TVector.
5. Набор автоматических тестов с использованием Google C++ Testing Framework.

# Руководство пользователя

Рассмотрим пример использования классов TMatrix и TVector. Подробно рассмотрим только класс TMatrix, т.к. TVector проще и во многом очень похож на TMatrix.

При запуске программы на консоль будут выведены 2 автоматически созданные матрицы. Ниже будут расположены результаты выполнения операций сложения, вычитания и умножения между ними.

Затем пользователю будет предложено ввести матрицу с консоли. Сразу после ввода она будет выведена на экран. На этом работа программы прекращается.

# Руководство программиста

## Описание структуры программы

Программа состоит из следующих модулей:

* Модули *Vector* и *Matrix*. Содержат примеры реализаций программ с использованием классов вектор и матрица соответственно. Файлы с реализацией *Vector\_ main.cpp* и *Matrix­\_ main.cpp.*
* Модуль *VectorLib*. Содержит файл *Vector.h*, в котором определен интерфейс и реализация шаблонного класса вектор *TVector*.
* Модуль *MatrixLib*. Содержит файл *Matrix.h*, в котором определен интерфейс и реализация шаблонного класса матрица *TMatrix*.
* Модули *VectorTest* и *MatrixTest*. Содержат для каждого из классов (*TVector* и *TMatrix*) наборы тестов, реализованные в файлах *VectorTest.cpp* (22 теста) и *MatrixTest.cpp* (18 тестов) с помощью использования фреймворка Google Test.

## Описание структур данных

#### TVector

Класс *TVector* является шаблонным. Всего в классе 2 поля *int size* – размер вектора и *T\* vec* – указатель на область памяти для хранения вектора данных типа *T*. Поля класса объявлены со спецификатором private.

**Дружественные функции:**

*friend istream& operator>>(istream &is, TVector<FriendT> &V)* – ввод вектора через консоль. Принимает ссылку на стандартный поток ввода и ссылку на объект класса *TVector*, возвращает ссылку на стандартный поток ввода.

friend ostream& operator<<(ostream &os, const TVector<FriendT> &V) – вывод вектора на консоль. Принимает ссылку на стандартный поток вывода и ссылку на объект класса *TVector*, возвращает ссылку на стандартный поток вывода.

**Конструкторы и методы класса объявлены со спецификатором public:**

*TVector<T>(int n = 0)* – конструктор по умолчанию.

*TVector<T>(const TVector<T> &V)* – конструктор копирования.

*virtual ~TVector<T>()* – деструктор.

**Методы для работы с классом TVector:**

*int GetSize() const* – возвращает размер вектора.

*T& operator[](int i)* – 0-based индексация.

*bool operator==(const TVector<T> &V) –* проверка векторов на равенство. Возвращает true, если равенство выполняется, false в противном случае.

*bool operator!=(const TVector<T> &V)* – проверка векторов на неравенство. Возвращает true, если неравенство выполняется, false в противном случае.

**Перегрузка операторов:**

*TVector operator-(const T &val) –* вычесть из вектора скаляр. Создается временный вектор. Каждая координата исходного вектора уменьшается на данное число.

*TVector operator+(const T &val) –* прибавить к вектору скаляр. Создается временный вектор. Каждая координата исходного вектора увеличивается на данное число.

*TVector operator\*(const T &val)* – умножение вектора на число. Создается временный вектор. Каждая координата исходного вектора умножается на данное число.

*TVector& operator=(const TVector<T> &v)* – оператор присваивания одного вектора другому.

*TVector operator+(const TVector<T> &v)* – сложение векторов. Если размерности векторов совпадают, то создается временный вектор, куда записывается результат поэлементного сложения соответствующих координат двух векторов.

*TVector operator-(const TVector<T> &v)* – вычитание векторов. Если размерности векторов совпадают, то создается временный вектор, куда записывается результат поэлементного вычитания соответствующих координат двух векторов.

*T operator\*(const TVector<T> &v) –* скалярное произведение. Если размерности векторов совпадают, то создается временная переменная, в которую записывается сумма произведений соответствующих координат.

#### TMatrix

Класс TMatrix является шаблонным и является наследником TVector как «вектор векторов» : template <class T> class TMatrix : public TVector<TVector<T>

**Дружественные функции:**

*friend istream& operator>>(istream &in, TMatrix<FriendT> &MT)* – ввод матрицы через консоль. Принимает ссылку на стандартный поток ввода и ссылку на объект класса *TMatrix*, возвращает ссылку на стандартный поток ввода.

friend ostream& operator<<(ostream &out, const TMatrix<FriendT> &MT) – вывод матрицы на консоль. Принимает ссылку на стандартный поток вывода и ссылку на объект класса *TMatrix*, возвращает ссылку на стандартный поток вывода.

**Элементы класса, объявленные со спецификатором public:**

*TMatrix(int s = 10)* – конструктор с параметром.

*TMatrix(const TMatrix &MT)* – конструктор копирования.

*TMatrix(const TVector<TVector<T> > &MT)* – конструктор преобразования типа.

*virtual ~TMatrix<T>()* – деструктор.

**Перегруженные операторы:**

*bool operator==(const TMatrix &MT) –* оператор проверки на равенство.

*bool operator!=(const TMatrix &MT)* – оператор проверки на неравенство.

*TMatrix& operator=(const TMatrix &MT) –* операторприсваивания.

*TMatrix operator+(const TMatrix &MT) –* оператор сложения.

*TMatrix operator-(const TMatrix &MT)* – оператор вычитания.

*TMatrix operator\*(const TMatrix &MT)* – оператор умножения.

*TMatrix operator\*(const TMatrix &MT)* – оператор деления.

## Описание алгоритмов

1. **Перегрузка оператора умножения.**

Произведением матриц А и B является такая матрица C = AB, у которой элемент , стоящий в *i*-ой строке и *j*-ом столбце, равен сумме произведений элементов *i*-ой строки матрицы А на соответствующие элементы *j*-го столбца матрицы B.

Реализация перегрузки оператора умножения представлена, с помощью трех вложенных циклов: по строкам первой матрицы, по столбцам второй матрицы и по элементам текущего столбца второй матрицы.

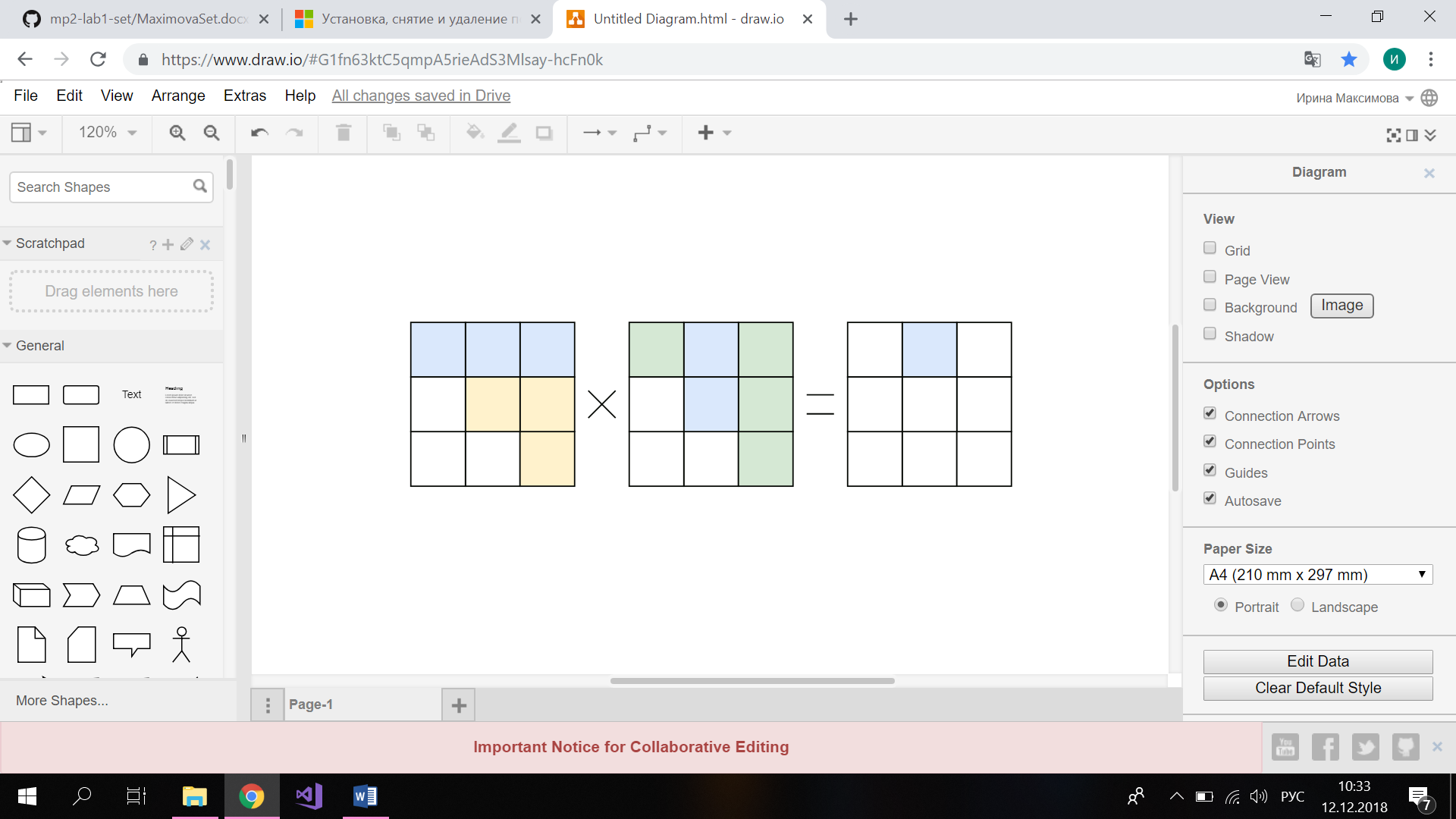


Рисунок 2. Умножение верхнетреугольных матриц

1. **Перегрузка оператора деления.**

Делением матриц D и A является такая матрица C = D\*, где - обратная матрица к матрице A. Обратная матрица может быть найдена методом Гаусса: Записываем расширенную матрицу :

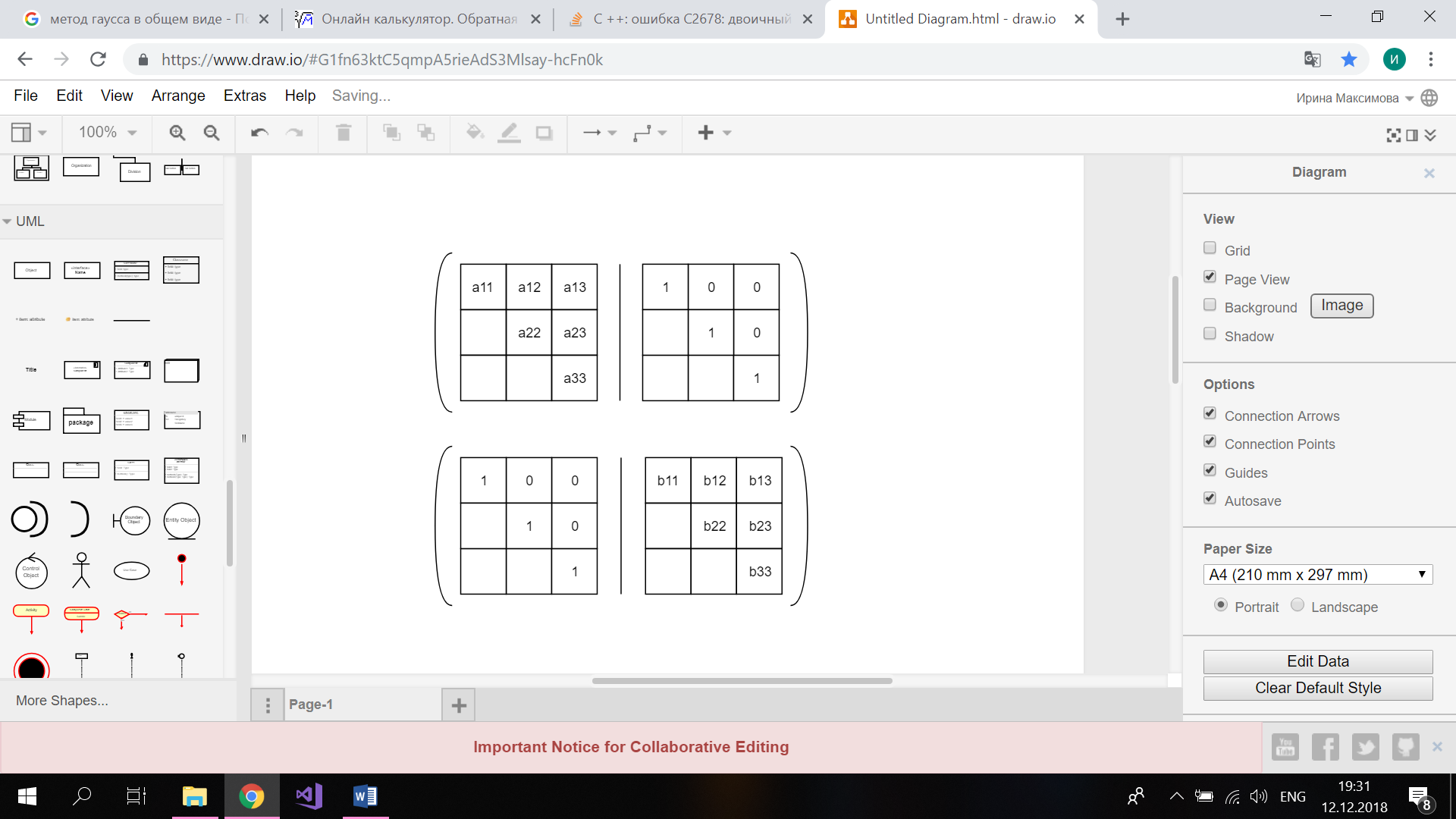


Рисунок 3. Расширенная матрица

Далее с помощью элементарных преобразований строк матрицы справа от черты получаем единичную матрицу, а то что получается справа и есть искомая обратная матрица :

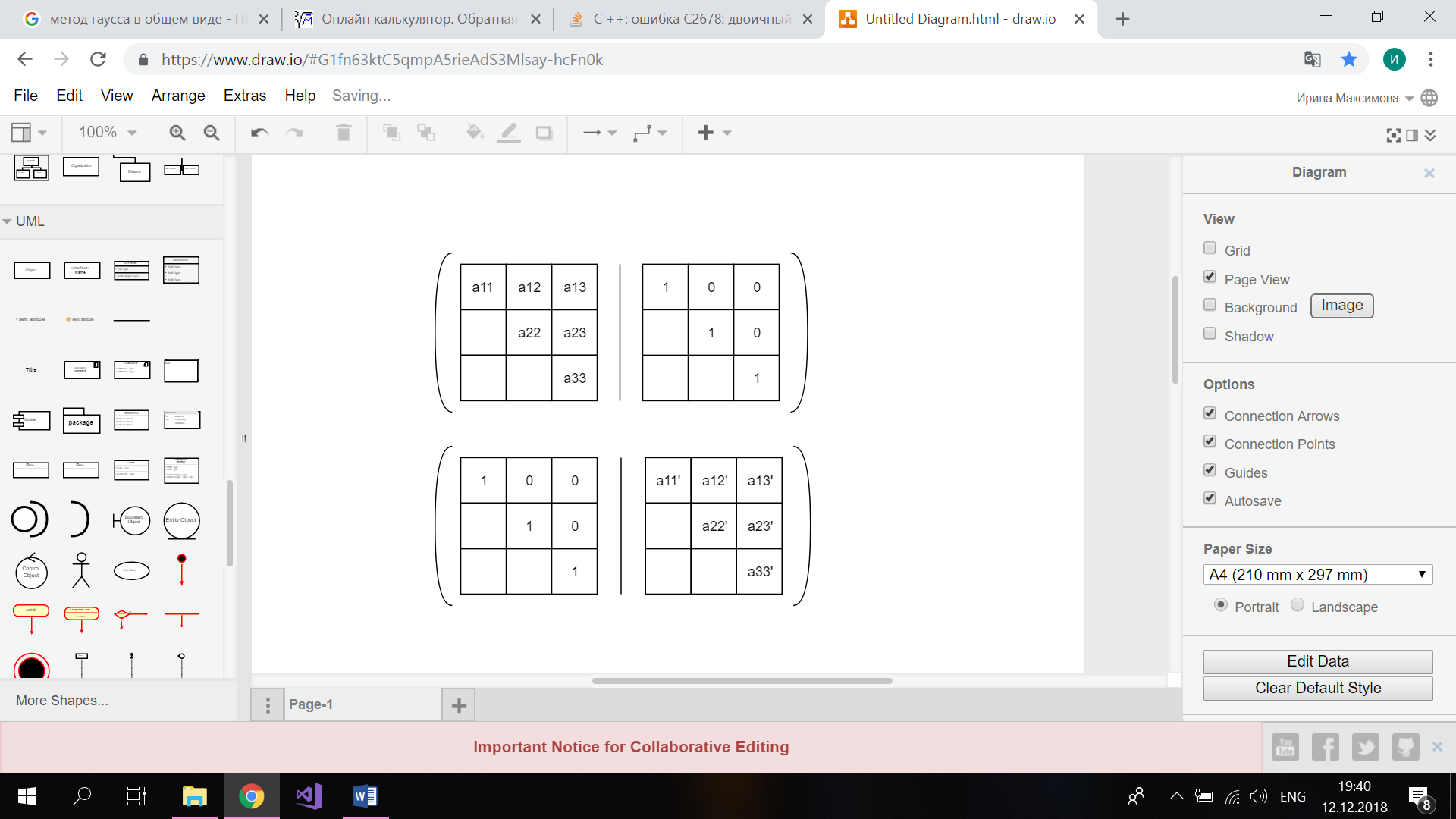


Рисунок 4. Конечная таблица метода Гаусса )

# Эксперименты

Эксперименты проводились на ПК с следующими параметрами:

1. Операционная система: Windows 10 Домашняя
2. Процессор: Intel(R) Core™ i5-7200U CPU @ 2.70 GHz
3. Версия Visual Studio: 2017

|  |  |  |
| --- | --- | --- |
| Кол-во элементов | Время работы оператора сложения (в млс)  O(n2) | Время работы оператора умножения (в млс)  O(n3) |
| 1000 | 35 | 9 886 |
| 5 000 | 844 | 1 263 525 |
| 10 000 | 3702 | 10 606 545 |

В работе использовались классические методы сложения и умножения матриц для которых сложность составляет соответственно O(n2) и O(n3). Тем не менее из таблицы видно, что программа работает чуть быстрее, чем ожидаемая цифра. Это достаточно легко объяснить.

Мы работаем исключительно с верхнетреугольными матрицами и храним только элементы выше главной диагонали, тем самым мы повышаем эффективность использования памяти почти в 2 раза. Вместо того чтобы хранить матрицу полностью используя память n2, мы храним только необходимую часть матрицы, занимая всего n\*(n+1)/2 памяти. За счет уменьшения объема используемой память и возрастет скорость работы программы, так как самой долгой операцией является операция доступа к данным.

# Заключение

В ходе выполнения лабораторной был произведен анализ задачи - установлено понятие верхнетреугольных матриц.

Осуществлено проектирование работы:

* Понятие верхнетреугольной матрицы
* Представление верхнетреугольной матрицы в виде вектора векторов
* Выделение базового класса для реализации класса матриц

В результате проделанной работы мне удалось

1. Реализовать класс битового поля TBitField.
2. Реализовать класс множества TSet.
3. Обеспечить работоспособность тестов и примера использования.
4. Реализовать нескольких тестов программы на базе Google Test.

Таким образом, благодаря этой лабораторной работе мне удалось разработать структуру хранения верхнетреугольных матриц.

# Литература

* Книги

1. A.O. Грудзинский. Методы программирования, Издательство Нижегородского госуниверситета, 2006.
2. Васильев А.Н. Самоучитель С++ с примерами и задачами. -СПб.: Наука и Техника, 2016. -480с.

* Ссылки в Internet

1. Гергель В.П. Методические материалы по курсу «Методы программирования 2»: [<http://www.itmm.unn.ru/files/2018/10/Primer-1.2.-Struktury-hraneniya-matrits-spetsialnogo-vida.pdf>], 2015.
2. Википедия: свободная электронная энциклопедия: на русском языке [https://ru.wikipedia.org/wiki/Матрица\_(математика)]
3. Справочный материал по математике, геометрии и физике [Электронный ресурс] [<http://ru.solverbook.com/spravochnik/matricy/umnozhenie-matric/>]