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

Денисов Владислав Львович

\_\_\_\_\_\_\_\_\_\_\_ Подпись

**Научный руководитель:**

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

Нижний Новгород

2018.

Содержание

[1. Введение 3](#_Toc532664617)

[2. Постановка задачи 4](#_Toc532664618)

[3. Руководство пользователя 5](#_Toc532664619)

[4. Руководство программиста 7](#_Toc532664620)

[4.1 Описание структуры программы 7](#_Toc532664621)

[4.2 Описание структур данных 7](#_Toc532664622)

[4.3 Описание алгоритмов 9](#_Toc532664623)

[4.4 Оценка сложности алгоритмов 12](#_Toc532664624)

[5. Заключение 14](#_Toc532664625)

[6. Литература 15](#_Toc532664626)

# 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.

Матричные обозначения широко распространены в современной математике и её приложениях. Ведь матрица – полезный инструмент для работы со многими задачами теоретической и прикладной математики. Так, одной из важнейших является задача нахождения решения систем линейных алгебраических уравнений.

Помимо матриц общего вида (прямоугольных), для которых наиболее естественной и 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.

В данной работе будут рассмотрены верхнетреугольные матрицы, которые в общем виде представляют собой квадратные матрицы, элементы которой ниже главной диагонали равны нулю.

# Постановка задачи

В рамках лабораторной работы ставится задача эффективной реализации структуры данных, представляющей собой верхнетреугольную матрицу. А также разработка методов, позволяющих в определенной мере работать с векторами и матрицами, представленными с помощью шаблонных классов.

Для этого необходимо реализовать перегрузку операторов у векторов:

* 0-based индексация,
* присваивание одного вектора другому,
* унарное сложение и вычитания,
* бинарное сложение и вычитание векторов,
* скалярное произведение векторов,
* умножение вектора на число и числа на вектор,
* проверка на равенство и неравенство,
* постфиксный и префиксный инкремент и декремент,
* ввод/вывод с помощью консоли.

В свою очередь, для матриц будет представлена перегрузка:

* присваивание одной матрицы другой,
* сложения и вычитания двух матриц,
* умножения двух матриц,
* деления двух матриц,
* проверка на равенство и неравенство,
* ввод/вывод с помощью консоли.

Программное решение будет выглядеть следующим образом:

1. Вспомогательный класс для создания верхнетреугольных матриц TVector.
2. Класс матриц TMatrix, реализованный с использованием вспомогательного класса TVector.
3. Класс 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. Однако 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.

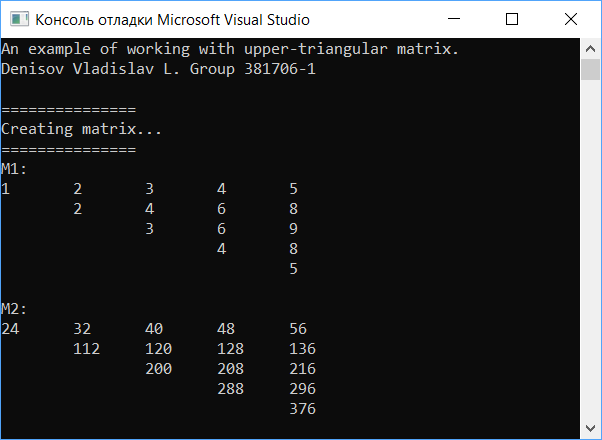


Рисунок 1 Автоматически созданные матрицы.

Ниже будут расположены результаты выполнения операций сложения, вычитания, умножения и деления между ними.

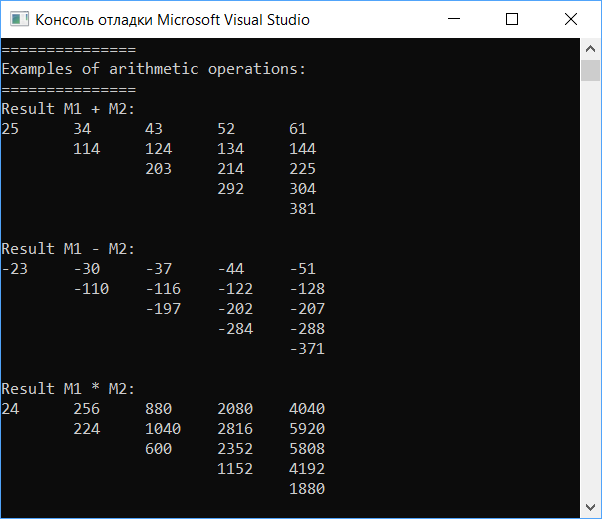


Рисунок 2 Результаты выполнения арифметических операций.

Затем пользователю будет предложено ввести матрицу с консоли. Будем следовать представленной инструкции и выполним необходимые действия. Сразу после ввода она будет выведена на экран.

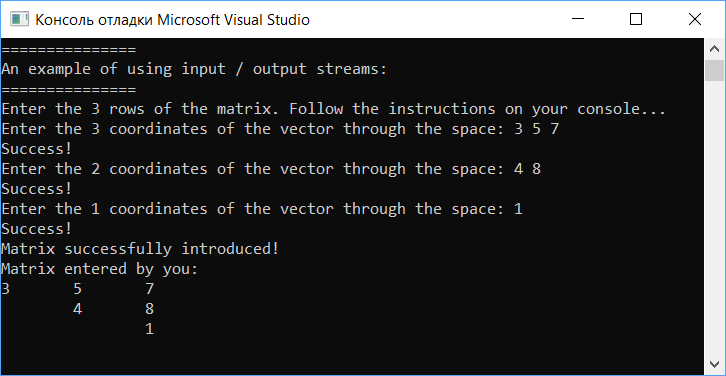


Рисунок 3 Пример работы операторов ввода/вывода.

Производим попытку сложения автоматически созданной матрицы размера 5 и введенной нами размера 3. Получаем сообщение об ошибки – было вызвано исключение.

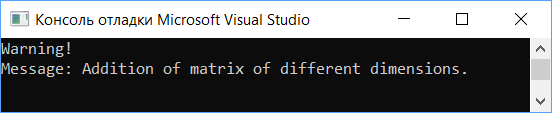


Рисунок 4. Пример обработанного исключения.

# Руководство программиста

## 4.1 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 *main\_Vector.cpp* и *main\_Matrix.cpp.*
* Модуль *VectorLib*. Содержит файл *Vector.h*, в котором определен соответственно интерфейс шаблонного класса вектор *TVector*. И по причине того, что класс шаблонный – там же находится реализация его методов.
* Модуль *MatrixLib*. Содержит файл *Matrix.h*, в котором определен соответственно интерфейс шаблонного класса матрица *TMatrix*. И по причине того, что класс шаблонный – там же находится реализация его методов.
* Модули *VectorTest* и *MatrixTest*. Содержат для каждого из классов (*TVector* и *TMatrix*) наборы тестов, реализованные в файлах *test\_vector.cpp* (23 теста) и *test\_matrix.cpp* (20 тестов с помощью использования фреймворка Google Test

## Описание структур данных

#### Класс TVector – вектор.

Рассмотрим класс *TVector* подробно.

template <class T> class TVector {…} – класс является шаблонным.

**Дружественные функции:**

*template <class FriendT> friend TVector<FriendT> operator\*(FriendT a, const TVector<FriendT> &V)* – умножение числа на вектор. Реализация аналогично методу умножения вектора на число, описание ниже.

*template <class FriendT> 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.

template <class FriendT> 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.

**Элементы, объявленные со спецификатором private**:

*int size* – размерность вектора.

*T \*mas* – указатель на область памяти для хранения вектора.

**Со спецификатором public:**

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

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

*virtual ~TVector<T>()* – деструктор. Освобождает выделенную под вектор память.

**Методы для работы с классом TVector:**

*int GetSize() const* – возвращает размерность 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 TVector<T> &other)* – оператор присваивания одного вектора другому.

*TVector operator++()* – префиксный инкремент.

*TVector operator++(int)* – постфиксный инкремент.

*TVector operator--()* – префиксный декремент.

*TVector operator--(int)* – постфиксный декремент.

*TVector operator+() const* – унарное сложение. Создается копия исходного вектора и возвращается без каких-либо изменений.

*TVector operator-()* const – унарное вычитание. Создается копия исходного вектора, каждая координата изменяет свой знак, затем копия возвращается.

*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.

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

#### Класс TMatrix – верхнетреугольная матрица.

Рассмотрим класс *TMatrix* подробно.

template <class T> class TMatrix : public TVector<TVector<T> > {…} – – класс матриц является шаблонным, является наследником TVector как «вектор векторов».

**Дружественные функции:**

*template <class FriendT> 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.

template <class FriendT> 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. В явном виде ничего не удаляет. Служит для вызова деструктора родительского класса *TVector.*

**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

Рассмотрим некоторые алгоритмы, работа которых не очевидна на первый взгляд.

**Создание матрицы. Конструктор с параметром.**

Вспомним, что матрица в нашей программе представляет собой вектор шаблонных векторов. Для этого конструктор класса *TMatrix* вызывает конструктор класса *TVector*, передавая в качестве шаблона *<T>* – *<TVector>*. Кроме этого действия для создания матрицы конструктор принимает ее размерность. Полученное значение проверяется на не отрицательность. В случае, если размерность выходит за допустимые границы, то возбуждается исключение. Если все хорошо, то в цикле заполняем каждую из ячеек созданного вектора векторов. Первая ячейка – вектор с размерностью матрицы. Затем в каждой ячейке создаем вектор с размерностью на единицу меньше, чем у предыдущего. В итоге получаем треугольную матрицу.

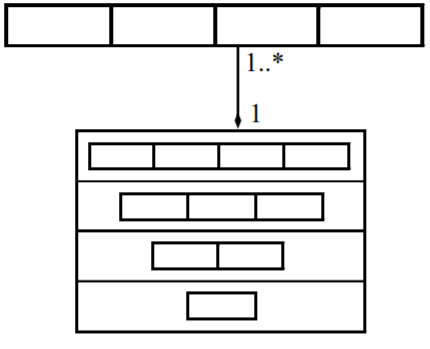
Шаблонный вектор.

Рисунок 1 Представление матрицы.

Матрица, построенная как вектор векторов.

**Конструкторы копирования и преобразования типа.**

Для того, чтобы воспользоваться конструктором копирования, необходимо просто вызвать конструктор копирования для вектора векторов, которому в качестве параметра передаем копируемую матрицу (в коде: *TMatrix(const TMatrix<T> &MT) : TVector<TVector<T> >(MT)).* Такой способ будет работать, так как *TMatrix* полностью создается на основе *TVector*, у которого в качестве шаблона используется *<T> – <TVector>.*

Конструктор преобразования типа работает аналогичным образом. В случае, если у нас имеется вектор векторов, то мы можем просто представить его в виде объекта класса *TMatrix.* Следовательно достаточно вызвать конструктор копирования для вектора векторов, как и в предыдущем случае (в коде: *TMatrix(const TVector<TVector<T> > &MT) : TVector<TVector<T> >(MT)*)

**Проверка на равенство и неравенство, перегрузка операторов сложения и вычитания.**

Для сравнения матриц на равенство необходимо выполнить проверку на равенство вектора векторов. Следовательно достаточно вызвать аналогичные операторы для *TVector* и вернуть результат их работы (в коде: *TVector<TVector<T> >::operator==(MT)* и *TVector<TVector<T> >::operator!=(MT)*).

Для перегрузки операторов сложения и вычитания требуется сложить/вычесть соответствующие элементы каждой из матриц. Снова на помощь приходит то, что матрица представлена как *TVector <TVector>*. Когда мы будем складывать элементы строки одной матрицы с элементами строки другой матрицы – мы будем выполнять сложение соответствующих векторов. И так для каждой строки. Следовательно, при выполнении аналогичных операторов для *TVector* *<TVector>* будет получен верный результат и для *TMatrix*  (в коде: *TVector <TVector<T> > :: operator+(MT)* и *TVector <TVector<T> > :: operator-(MT)*).

**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.

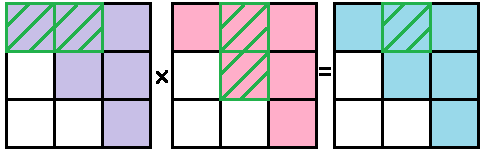


Рисунок 2 Умножение матриц.

**Перегрузка оператора деления**

Результатом деления 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 :

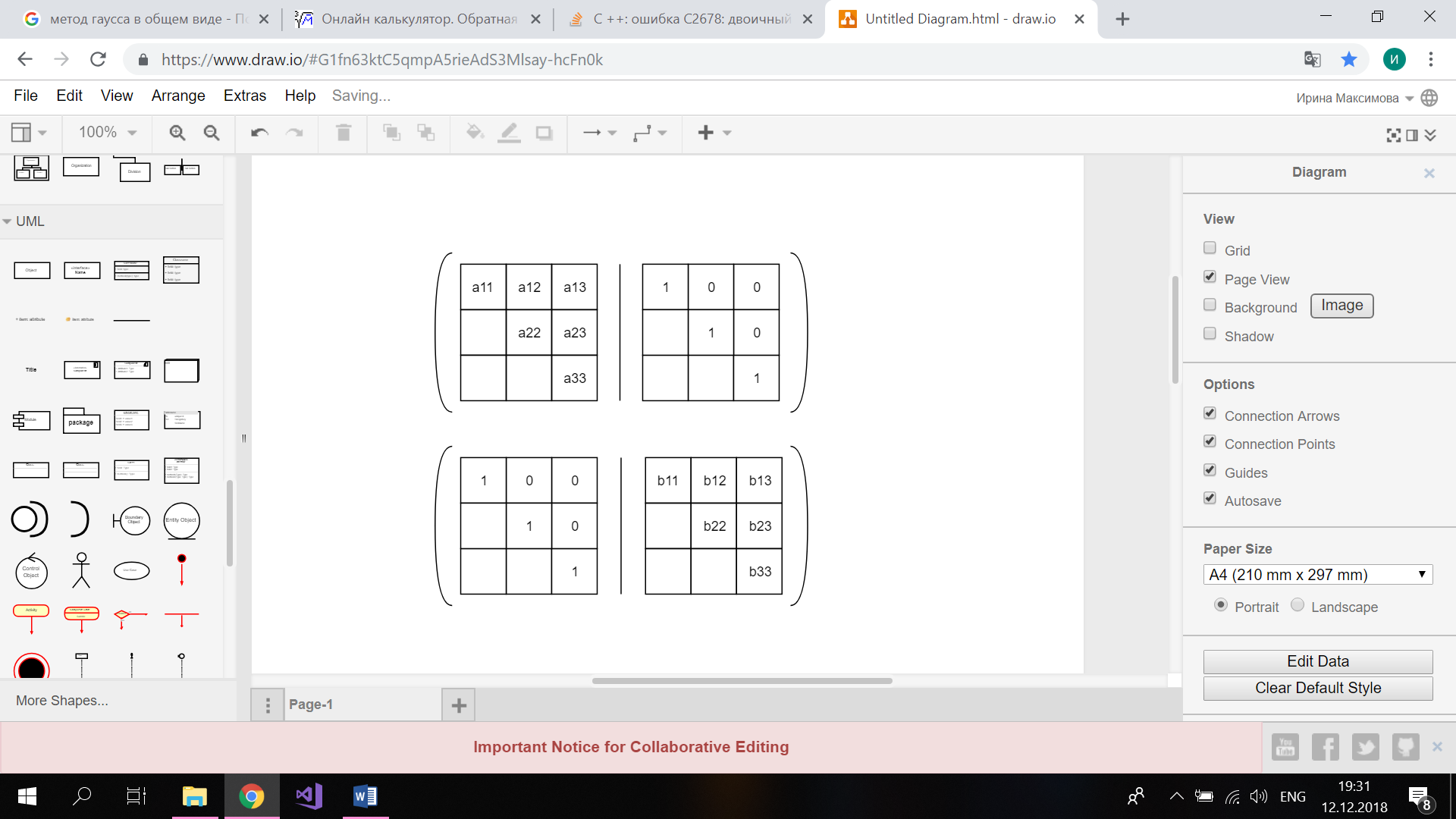


Рисунок 3 Расширенная матрица

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

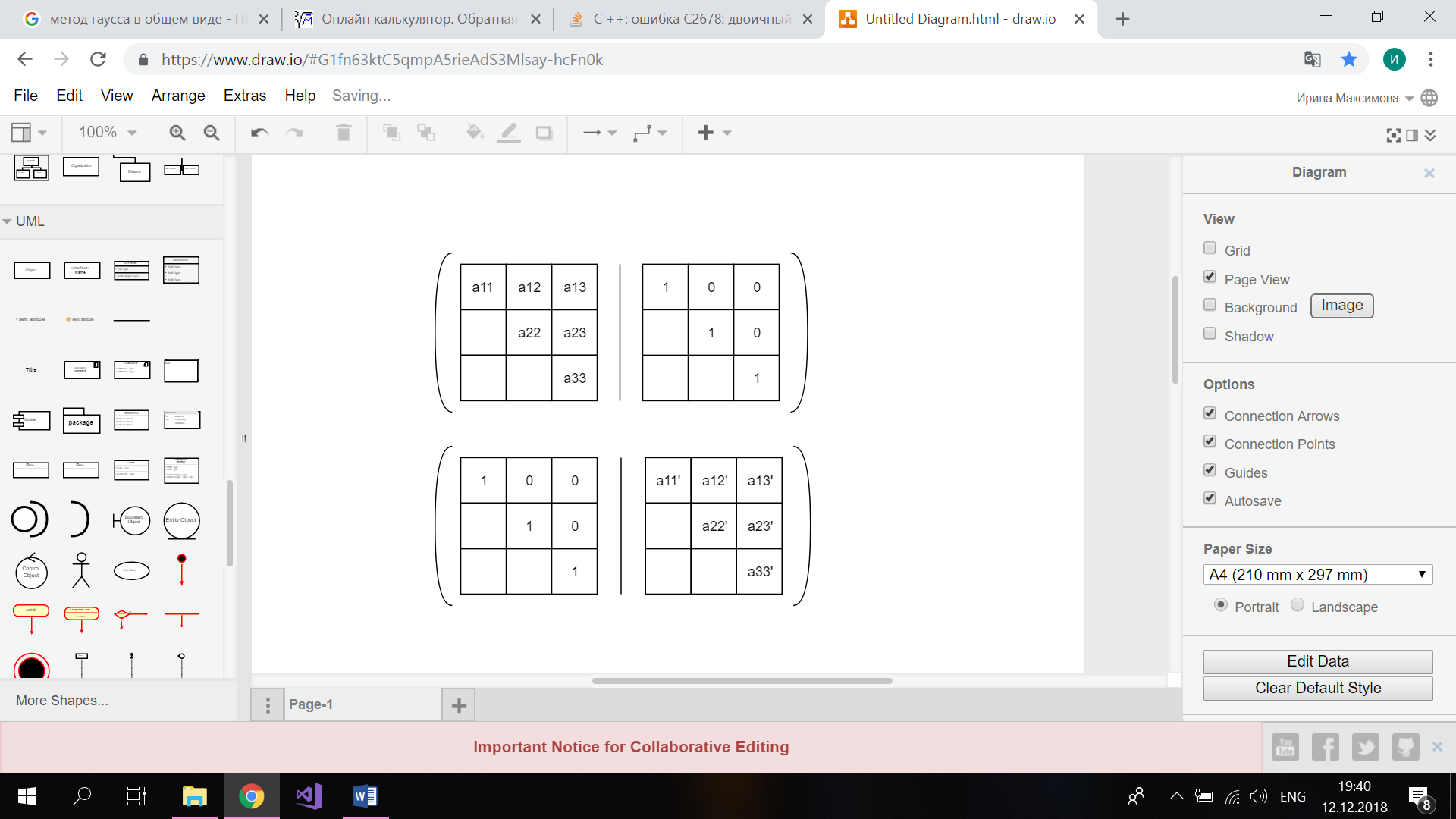


Рисунок 4 Конечная таблица метода Гаусса ).

## Оценка сложности алгоритмов

График 1 Зависимость времени работы алгоритма умножения двух матриц от числа элементов.

Проанализировав график 1 зависимости времени выполнения умножения двух верхнетреугольных матриц от размера матрицы, а также из анализа непосредственно алгоритма умножения, можем сделать вывод, что его сложность составляет .

График 2 Зависимость времени работы алгоритма сложения двух матриц от числа элементов.

Выполнив аналогичные действия для алгоритма сложения и графика 2, можем сделать вывод о том, что его сложность находится в районе .

# Заключение

В результате лабораторной работы была разработана библиотека, позволяющая в определенной мере работать с векторами и матрицами, представленными с помощью шаблонных классов. А именно, выполнять различные операции над ними, задача реализации которых была поставлена в начале данной лабораторной работы.

Были разработаны и доведены до успешного выполнения тесты, разработанные для данного программного проекта с использованием Google C++ Testing Framework.

Программное решение было продемонстрировано с помощью простейшего набора операций над матрицами. Пример был описан в разделе «Руководство пользователя».

# Литература

1. Википедия: свободная электронная энциклопедия: на русском языке [Электронный ресурс] // URL: https://ru.wikipedia.org/wiki/Матрица\_(математика) (дата обращения: 09.12.2018)
2. Методы программирования [Электронный ресурс] // URL: <http://www.itmm.unn.ru/files/2018/10/Primer-1.2.-Struktury-hraneniya-matrits-spetsialnogo-vida.pdf> (дата обращения: 10.12.2018)
3. Справочный материал по математике, геометрии и физике [Электронный ресурс] // URL: <http://ru.solverbook.com/spravochnik/matricy/umnozhenie-matric/> (дата обращения: 10.12.2018)