Федеральное агентство по образованию Российской Федерации

Государственное образовательное учреждение

высшего профессионального образования

Нижегородский государственный университет им. Н.И. Лобачевского

Институт информационных технологий математики и механики

Направление подготовки: «Фундаментальная информатика и информационные технологии»

Отчёт по лабораторной работе

**Разработка структуры хранения класса верхнетреугольных матриц**

Выполнила:

студентка ИИТММ гр. 381906-2

Гордей М.В.

Проверила:

программист каф. МОСТ

Усова М. А.

Нижний Новгород

2020 г.

Содержание

I. [Введение 3](#_Toc24457830)

II. [Постановка задачи](#_Toc24457831) 4

III. [Руководство пользователя](#_Toc24457832) 5

IV. [Руководство программиста](#_Toc24457833) 6

1. [Описание структуры программы](#_Toc24457834) 6

2. [Описание структур данных](#_Toc24457835) 7

3. [Описание алгоритмов](#_Toc24457836) 7

V. [Эксперименты](#_Toc24457837) 9

VI. [Заключение](#_Toc24457838) 10

VII. [Литература](#_Toc24457839) 11

VIII. [Приложение 1](#_Toc24457840)2

# Введение

Матрицы и векторы – важнейшие понятия в курсе линейной алгебры. Матрица в математическом понимании – объект, записываемый в виде прямоугольной таблицы, которая представляет собой совокупность строк и столбцов, на пересечении которых находятся её элементы. Векторами называются матрицы, состоящие из одной строки или одного столбца.

Чаще всего в математике работают с матрицами общего вида, но также в математических приложениях используются матрицы специального вида – треугольные, диагональные.

В этой работе созданы классы для работы с матрицами и векторами. Верхнетреугольные матрицы - квадратные матрицы, элементы которой ниже главной диагонали равны нулю.

# 2. Постановка задачи

Написать классы для работы с векторами и матрицами, использовать шаблоны.

Продемонстрировать корректную работу на примере (проверить правильность выполнения операций, проверка свойств матрицы)

Создание программных средств для эффективного хранения матриц верхнетреугольного вида, выполнение основных операций как сложение/вычитание умножение, копирование, сравнение. Требования к матрицам : число строк равняется числу столбцов. В процессе выполнения лабораторной работы требуется использовать систему контроля версий Git и фрэймворк для разработки автоматических тестов Google Test. Сделать выводы о проделанной работе.

В данной лабораторной работе должны быть реализованы : конструкторы( по умолчанию, инциализации, копирования), деструктор, должен быть доступ к защищенным полям, перегружены основные операции с векторами и матрицами.

В экспериментальной части необходимо проверить корректность работы программы.

# 3. Руководство пользователя

Шаги, совершаемые пользователем:

1. Зайти в репозиторий - <https://github.com/gordushha/MatrixLab>
2. Выбрать нужную ветку – Lab 1
3. Скачать код программы. Скачаем код архивом. Для этого нужно нажать на кнопку “Code”(справа вверху), затем на кнопку “Download ZIP”. После этих действий появится окно проводника. Нужно сохранить данный файл на рабочий стол и извлечь из него файлы программы с помощью архиватора (нажать правой кнопкой мыши на файл, выбрать команду “Извлечь все”).

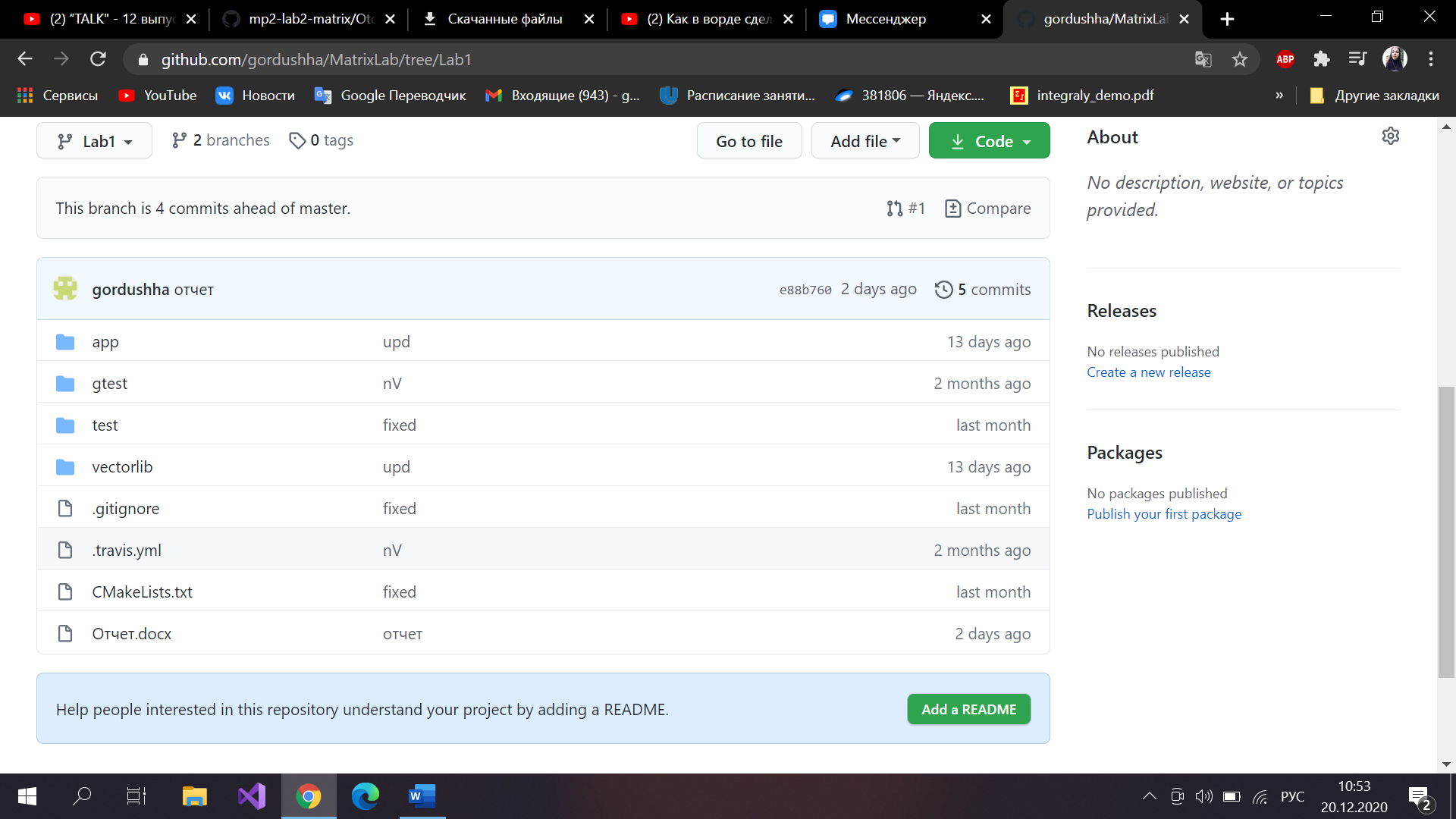


Рисунок 1. Скриншот ветки репозитория на GitHub

4) Открыть папку с программой на рабочем столе

5) Создать папку «build»

6) Перейти в эту папку, нажать правой кнопкой мыши в поле проводника, нажать «Git Bush Here» (предварительно скачать Git с официального сайта)

7) Благодаря действиям из предыдущего пункта на экране появится консоль, в которой нужно прописать команду «cmake ../»(предварительно нужно скачать программу CMake позволяющую автоматически собрать программу из исходного кода

8) Теперь проект Matrix можно запустить в Visual studio (пользвателю необходимо предварительно скачать эту программу)

9) Пользователю нужно нажать на проект main , и кликнув правой кнопкой выбрать «Назначить автозагружаемым проектом» ( при русской версии программы)

10) Затем нужно нажать сочетание клавиш CTRL+F5 на клавиатуре. Откроется консольное приложение, демонстрирующее работу матриц и векторов.

# 4. Руководство программиста

## 4.1. Описание структуры программы

Программа состоит из нескольких блоков:

ALL\_BUILD, gtest, ZERO\_CHECK - вспомогательные записи, ZERO\_CHECK перезапустит cmake. ALL\_BUILD - это цель, которая строит весь проект в активное решение

В блоке vectorlib содержится шаблонный класс Vector, объявлены все методы и их определения класса векторов, и его наследник My Matrix, в котором объявлены все методы класса матриц и их определения, объявлены и реализованы функции перегрузки операторов матрично-векторного умножения

В блоке app содержится один файл main с примером работы с классами класс Vector, Matrix

Также есть блок test, в которых содержится набор тестов, с помощью которых мы проверяем корректную работу программы

**4.2 Описание структур данных**

В программе определены два следующих шаблонных класса:

Class Vector,

Class Matrix

внутри класса Vector определены следующие protected поля:

* T \*x – шаблонный указатель;
* Int length длина вектора;
* Int StartIndex– стартовая ячейка.

внутри класса Matrix определен следующий набор public-методов

(плюс конструкторы и деструктор):

* Vector() – конструктор по умолчанию, не принимает никаких параметров, инициализирует все поля 0;
* Vector(Vector<T> & \_v) – конструктор копирования, принимает на вход объект типа TVector, создает объект с теми же характеристиками что и переданный, и массивом того же содержимого;
* Vector() – деструктор, очищает память;
* Int Lenght () – метод возвращающий длину вектора;
* void SetStartIndex () – метод, устанавливающий стартовый индекс;
* T operator [](const int index) – перегрузка оператора индексации, позволяет возвращать элемент вектора по индексу;
* Vector<T> operator+(const Vector<T>& \_v) – перегрузка оператора суммы, позволяет возвращать вектор, являющийся суммой двух векторов;
* Vector<T> operator-(const Vector<T>& \_v) – перегрузка оператора разности, позволяет возвращать вектор, являющийся разностью двух векторов;
* Vector<T> operator = (const Vector<T>& \_v) – перегрузка оператора присвоить, позволяет присваивать один объект типа Vector другому;
* bool operator ==(const Vector<T>& \_v) – перегрузка оператора сравнения, позволяет сравнивать объекты типа Vector;
* bool operator !=(const Vector<T>& \_v) – перегрузка оператора не равно, true Если операнды имеют одинаковое значение; в противном случае возвращается false
* T operator\*(const Vector<T>& \_v) – перегрузка оператора умножить, позволяет перемножать объекты типа Vector;
* friend ostream& operator<<(std::ostream& stream, const Vector<T1>& A)– перегрузка оператора вывода, позволяет выводить вектора на экран;
* friend istream& operator>>(std::istream& stream, Vector<T1>& A) – перегрузка оператора ввода, позволяет вводить вектора.

шаблонный класс Matrix – публичный наследник класса Vector, полей в нем не определено.

* Matrix(const TMatrix&m) – конструктор копирования;
* Matrix(Vector<Vector<T> >&m) конструктор с параметром вектор ;
* Bool operator ==(const Matrix<T>&m) – перегрузка оператора равно, позволяет сравнивать объекты типа Matrix;
* Bool operator =(const Matrix&)const – перегрузка оператора присваивание;
* Matrix operator +(const Matrix&m) – перегрузка оператора суммы;
* Matrix operator -(const Matrix&m) - перегрузка оператора разности;
* Matrix operator \*(const Matrix&m) - перегрузка оператора умножить;
* friend ostream& operator <<(ostream&, Matrix<T1>&A) – перегрузка оператора вывода;
* friend istream& operator >>(istream&, Matrix<T1>&A) - перегрузка оператора ввода.

## 4.3 Описание алгоритмов

*Алгоритм суммирования или вычитания векторов*

Сначала выполняется проверка на совпадение размерности. Для сложения векторов необходимо сложить соответствующие координаты этих векторов. Создается вектор для записи результата. По циклу соответствующие координаты складываются или вычитаются, в результате работы возвращается результирующий вектор.

*Алгоритм умножения матриц*

Если чисто столбцов в первой матрице совпадает с числом строк во второй, то эти матрицы можно перемножить. На входе выполняется эта проверка. Создается матрица для записи результата. В цикле каждая строка первой матрицы почленно умножается с каждым столбцом второй матрицы. Возвращается результирующая матрица.

Алгоритмы нахождения произведения вектора на матрицу и матрицы на вектор аналогичны алгоритму умножения матриц.

## 

# 5. Проверка теоретической основы

Проверим корректность выполнения нашей программы. Проверим свойства верхнетреугольных матриц.

Найдем определитель матрицы.Определитель верхнетреугольной матрицы равен произведению чисел, стоящих на главной диагонали. Проверим, корректны ли вычисления определителя в работе нашей программы.

Зададим две матрицы, посчитаем их определитель

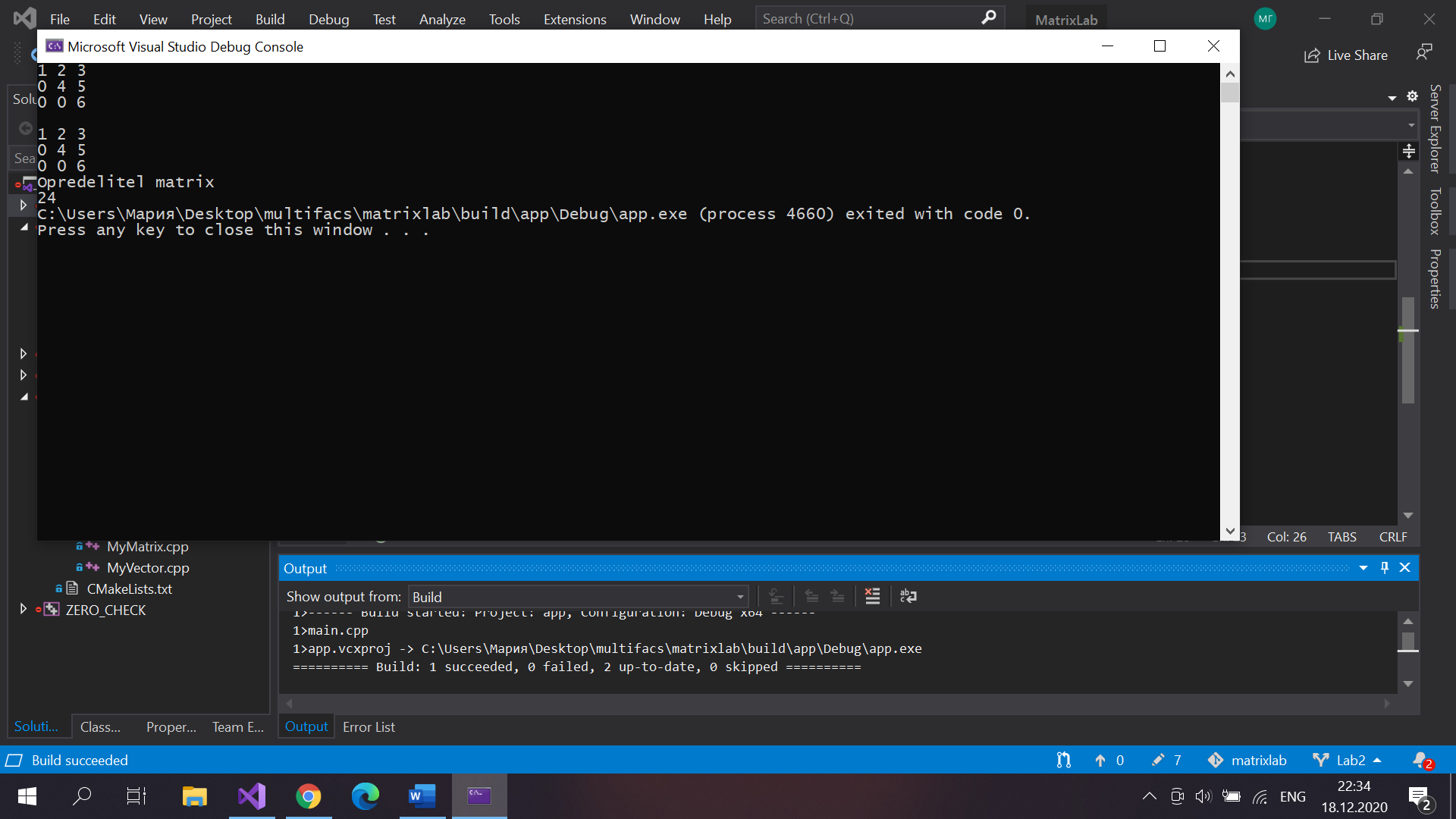


Рисунок 2: подсчет определителя

Перемножая элементы на главной диагонали видим, что программа работает корректно.

Проверим результат работы умножения матриц.

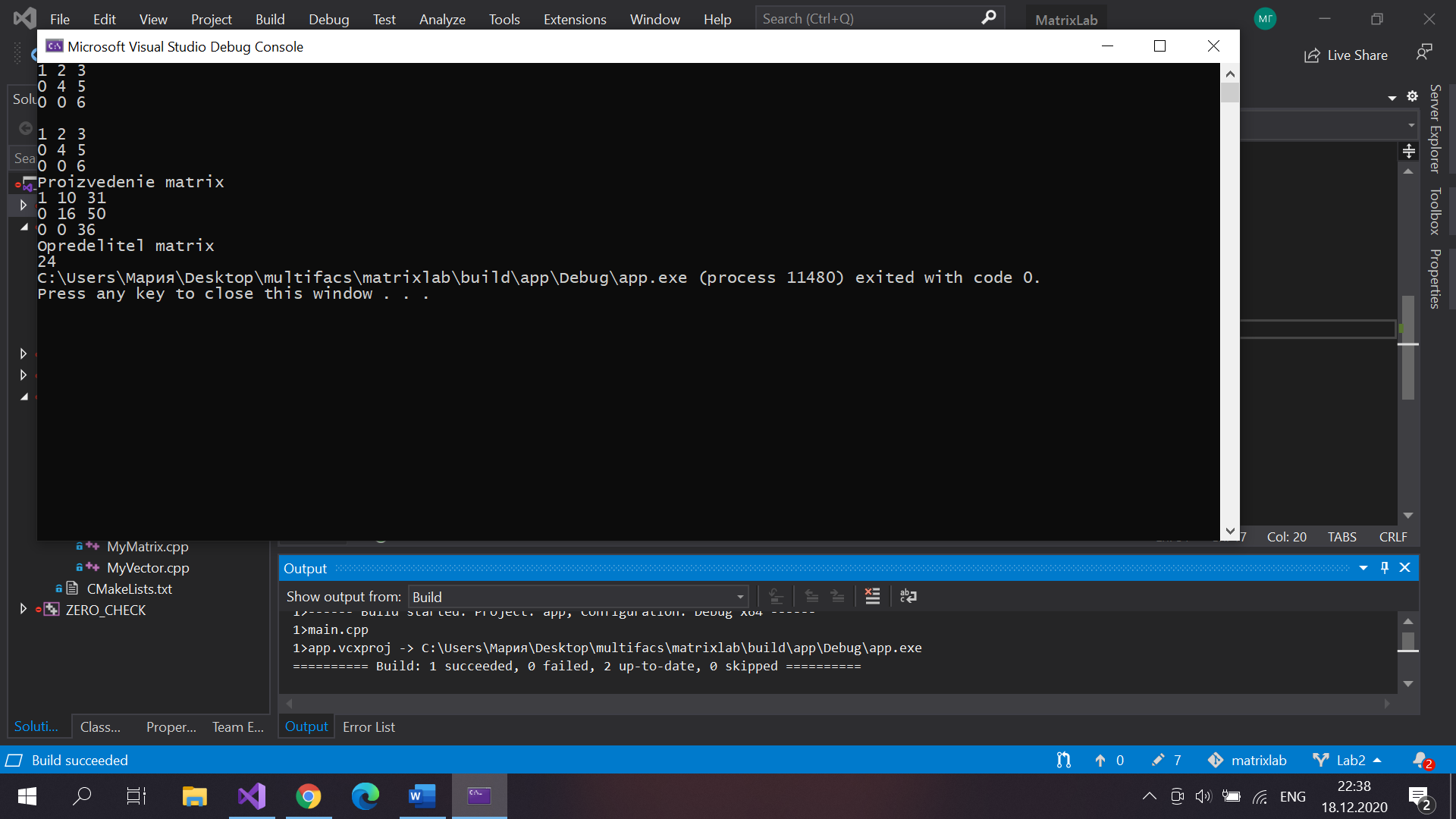


Рисунок 3: вычисление произведения двух матриц

Программа вычисляет произведение корректно.

# 6. Заключение

По итогу выполнения данной работы были написаны классы, позволяющие осуществлять удобную работу с матрицами и векторами.

Были переопределены арифметические операции для работы с матрицами, реализованными в данной структуре, проведены эксперименты, подтверждающие корректную работу программы. Также работоспособность программы проверена на тестах.

# 7. Литература

1. Официальный сайт Wikipedia. — Режим доступа <https://ru.wikipedia.org>
2. Официальный сайт Habr. — Режим доступа <https://habr.com>
3. Керниган, Б.У. Язык программирования С / Б.У. Керниган, Д.М. Ритчи

# 8. Приложение

Умножение треугольных матриц

Matrix<T> operator \*(Matrix<T>& \_m)

{

if (this->length != \_m.Length())

throw "Can't multiply";

Matrix<T> result(this->length);

T t;

for (int i = 0; i < this->length; i++)

for (int j = 0; j < this->length - i; j++)

{

t = 0; // i = 1; j = 1;

for (int k = i; k < j + i + 1; k++)

t += x[i].x[k - i] \* \_m[k].x[j - k + i];

result[i].x[j] = t;

}

Алгоритм сложения векторов:

template <class T>

Vector<T> Vector<T>::operator +(Vector<T>& \_v)

{

Vector<T> res;

res.length = MIN(length, \_v.length);

res.x = new T[res.length];

for (int i = 0; i < res.length; i++)

{

res.x[i] = x[i] + \_v.x[i];

}

return res;

}

Сложение матриц:

Matrix<T> operator +(Matrix<T>& \_m)

{

if (this->length != \_m.Length())

throw "Matrices' sizes aren't equal";

Matrix<T> result(this->length);

for (int i = 0; i < this->length; i++)

result[i] = this->x[i] + \_m[i];

return result;

}