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

**Реализация классов для работы с матрицами и векторами**

Выполнил:

студент ИИТММ гр. 381906-2

Кулемин П. А.

Проверил:

программист каф. математического обеспечения

и суперкомпьютерных технологий

Усова М. А.

Нижний Новгород

2020г.

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) 10

VI. [Заключение](#_Toc24457838) 11

VII. [Литература](#_Toc24457839) 12

VIII. [Приложение 1](#_Toc24457840)3

# 1. Введение

Матрицы и векторы, это важная часть векторной алгебры, и поэтому информатизация и автоматизация в этой области математики крайне важна. Именно для этого мной были созданы классы для работы с векторами и матрицами, а также произведена теоретическая и практическая оценка времени работы различных арифметических операций над матрицами.

# 2. Постановка задачи

Реализовать классы для работы с векторами и матрицами использовать шаблоны. Продемонстрировать их работу на примере.

Должны быть:

* конструкторы (по умолчанию, инициализатор, копирования).
* деструктор.
* доступ к защищенным полям.
* перегруженные операции: +, -, \*, =, ==, [].
* потоковый ввод и вывод.
* перегруженные операции +, -, \*, / должны быть реализованы для векторов (вектор +, -, \*, / вектор или скаляр), матриц (матрица +, -, \* матрица или скаляр).

Оценить асимптотически время работы матричных и векторных операций \* сравнить оценку с реальным временем работы. Сделать вывода.

# 3. Руководство пользователя

Шаги, совершаемые пользователем:

1. Зайти в интересуемый репозиторий на GitHub <https://github.com/George-Amberson/mp2-lab2-matrix>
2. Перейти в нужную ветку под названием Lab1
3. Скачать код программы. Это легко можно сделать, скачав файлы программы архивом. Для этого нужно нажать на кнопку “Code”, затем на кнопку “Download ZIP”. После этих действий появится окно проводника. Нужно сохранить данный файл на рабочий стол и извлечь из него файлы программы (нажать правой кнопкой мыши на файл, выбрать команду “Извлечь все”)

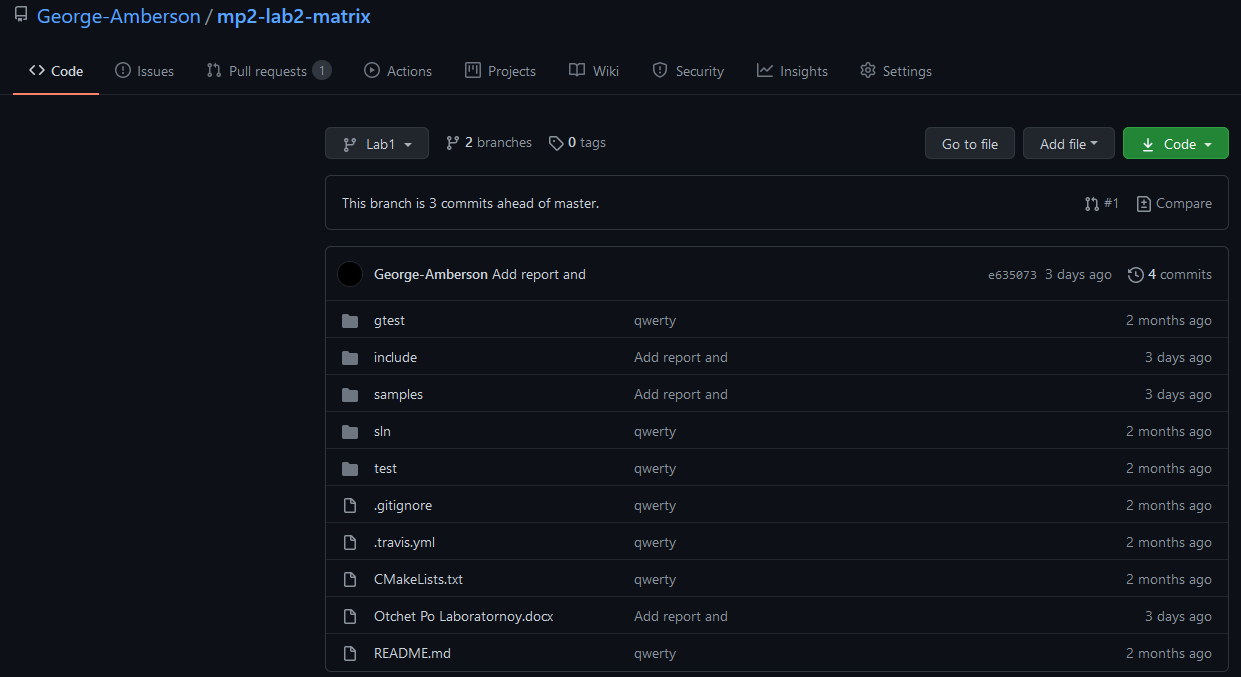


Рисунок 1.Скриншот ветки репозитория на GitHub

1. Открыть папку с программой на рабочем столе
2. Создать папку «build»
3. Перейти в эту папку, нажать ПКМ в поле проводника, нажать «Git Bush Here» (предварительно скачать Git с официального сайта)
4. 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
5. Теперь проект mp2-lab2-matrix можно запустить в Visual studio.
6. Пользователю нужно нажать на проект sample\_matrix, и через ПКМ выбрать «Назначить автозагружаемым проектом»
7. 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 6 решений.

В решении ALL\_BUILD, gtest, а также ZERO\_CHECK определен вспомогательный код, необходимый для работы тестовой системы и всей программы в целом.

В решении matrix содержится 1 модуль umatrix.h. В нем содержится объявление, а также определение шаблонного класса TVector, и его наследника TMatrix.

В решении sample\_matrix содержится один модуль, sample\_matrix.cpp содержащий в себе обьявление функции main с примером работы над объектами класса TVector и TMatrix.

В решении test\_matrix содержится 3 модуля: test\_main.cpp, test\_tvector.cpp, test\_tmatrix.cpp. В них содержится набор тестов, с помощью которых мы подвергаем нашу программу проверке на корректную работу.

## Описание структур данных

В программе определены два следующих шаблонных класса:

Class TMatrix

Class TVector

Внутри класса TVector определены следующие protected поля:

* ValType\* pvector– шаблонный указатель;
* int size– целочисленная длина вектора:
* int StartIndex– целочисленный номер стартовой ячейки для данного вектора;

Внутри класса Matrix определен следующий набор public-методов (плюс конструкторы и деструктор);

* TVector(int s = 10, int si = 0) – конструктор по умолчанию, и инциализатор в одномб принимает длину вектора и стартовый индекс;
* TVector(const TVector& v) – конструктор копирования, принимает на вход объект типа TVector, создает объект с теми же характеристиками что и переданный, и массивом того же содержимого;
* virtual ~TVector() – виртуальный деструктор, очищает выделенную конструкторами память;
* int GetSize () – метод возвращающий длину вектора;
* int GetStartIndex () – метод возвращающий стартовый индекс вектора;
* void SetSize(int) – метод устанавливающий размер вектора в соответствии с переданным аргументом;
* void SetStartIndex(int)- метод устанавливающий стартовый индекс в соответствии с переданным аргументом;
* VaType\*\* GetVector () – метод возвращающий указатель на блок памяти с вектором;
* ValType& operator[](int) – перегрузка оператора индексации;
* ValType& operator [ ](int) const - константная перегрузка оператора индексации;
* bool operator == (сonst TVector&) – перегрузка оператора равно;
* bool operator != (сonst TVector&) – перегрузка оператора неравно;
* TVector& operator = (TVector&) – перегрузка оператора присваивания;
* TVector operator +(const ValType& ) – перегрузка оператора сложения для объектов вектор и скаляр. Математически определяется как вектор с координатами, каждая из которых увеличена на скаляр;
* TVector operator -(const ValType&) - перегрузка оператора вычитания, по своей сути аналогична перегрузки сложения;
* TVector operator \*(const ValType&) - перегрузка оператора умножения на скаляр;
* TVector operator +(const TVector&) - перегрузка оператора сложения для двух векторов;
* TVector operator -(const TVector&) - перегрузка оператора вычитания, для двух векторов;
* ValType operator \*(const ValType&) - перегрузка оператора умножения, для двух векторов;
* int FindIndex() – возвращает номер первой ненулевой координаты вектора, или -1;
* friend ostream& operator <<(ostream& out, const TVector& rhs) – перегрузка оператора вывода, позволяет выводить вектора на экран;
* friend istream& operator >>(istream& in, TVector& rhs) - перегрузка оператора ввода, позволяет вводить вектора уже заданного размера, либо задавать размер и вводить самостоятельно.

Шаблонный класс TMatrix, является public, наследником класса TVector, при этом параметр шаблона определен как TVector<ValType> > поэтому никаких полей в этом классе не определено. Также почти 70% методов, он наследует от класса-родителя, поэтому внутри него, определены только специфичные для него методы:

* TMatrix(int s=10) –конструктор по умолчанию, и конструктор инциализатор ;
* TMatrix(const TMatrix&) – конструктор копирования;
* TMatrix(TVector<TVector<ValType> >&&) – move конструктор с параметром;
* TMatrix(TVector<TVector<ValType> >&) конструктор с параметром вектор с шаблоном вектор от шаблона;
* bool operator ==(const TMatrix&)const – перегрузка оператора равно;
* bool operator !=(const TMatrix&)const – перегрузка оператора неравно;
* TMatrix& operator =(const TMatrix&) – перегрузка оператора присваивания;
* TMatrix operator +(const TMatrix&) – перегрузка оператора суммы;
* TMatrix operator -(const TMatrix&) - перегрузка оператора разности;
* friend ostream& operator <<(ostream&, TMatrix&) – перегрузка оператора вывода н экран;
* friend istream& operator >>(istream&, TMatrix&) - перегрузка оператора ввода, с консоли.

## Описание алгоритмов

## Алгоритм суммирования или вычитания матриц:

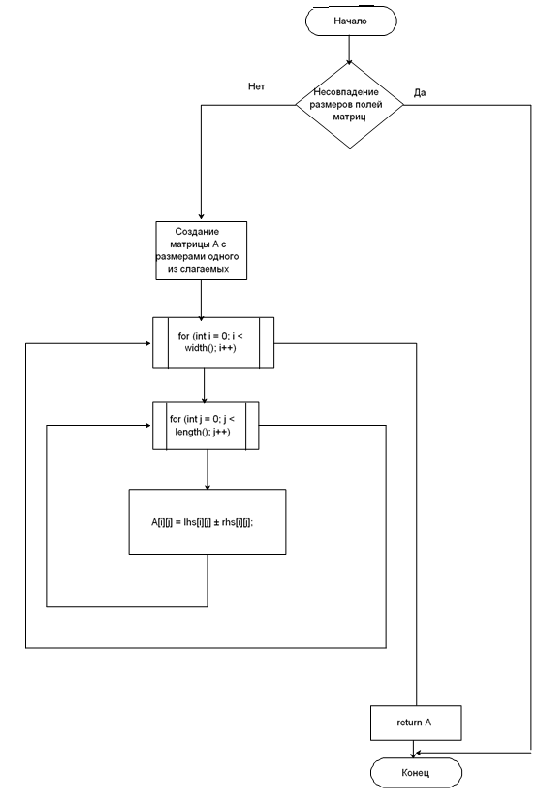


Рисунок 2: Блок-схема алгоритма сложения/вычитания векторов

Поскольку класс TMatrix это публичный наследник класса TVector, то для него нет смысла описывать каким-либо образом арифметическую операцию сложения, так как эта операция в числе других, наследуется от класса TVector.

## Алгоритм умножения двух векторов:

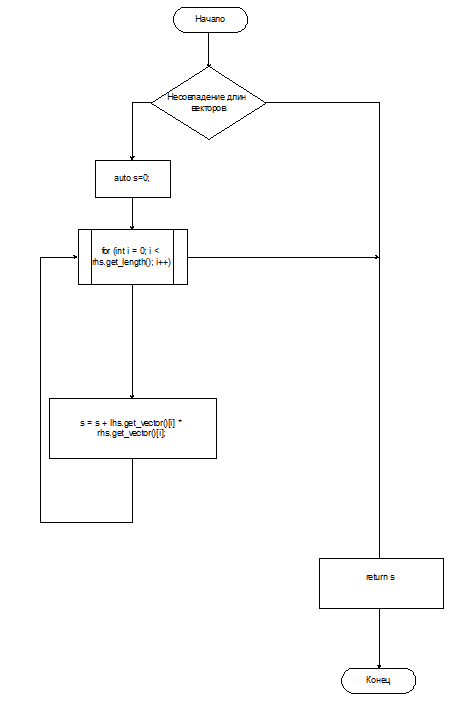


Рисунок 3: Блок-схема умножения двух векторов.

# Эксперименты

Оценим время, которое занимают векторные и матричные операции, с помощью

асимптотической сложности. Рассмотрим код отвечающий за суммирование матриц:

TMatrix<ValType>Res(\*this);

for (int i = Res.GetStartIndex(); i < Res.GetSize(); i++)

{

Res[i] = (\*this)[i] + mt[i];

}

Для векторов операция суммы и присваивания линейная значит асимптотическая сложность этого кода:

Теперь мы произведем замеры времени сложения квадратных матриц (чтобы точнее соотнести результаты замеров с асимптотической сложностью):

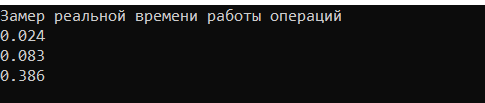


Рисунок 4: Замеры времени, полученные при выполнении сложения матриц (сек.).

На основе полученных данных, построим таблицу соотношения размеров матриц и времени их суммирования.

|  |  |
| --- | --- |
| Количество элементов в матрице | Время выполнения (сек.) |
| 2000 | 0.024 |
| 4000 | 0.083 |
| 8000 | 0.386 |

Таблица 1: Результаты замеров времени суммирования матриц.

Теперь оценим время работы векторного умножения с помощью асимптотической сложности. Рассмотрим код отвечающий за умножения вектора на вектор:

if (this->GetSize() != v.GetSize()) throw logic\_error("");

float res=0;

for (int i = 0; i < v.Size - v.StartIndex; i++)

{

res = res + v.pVector[i]\*this->pVector[i];

}

Для элементов вектора операция умножения работает за константу значит асимптотическая сложность этого кода:

Теперь мы произведем замеры времени сложения квадратных матриц (чтобы точнее соотнести результаты замеров с асимптотической сложностью):



Рисунок 5: Замеры времени, полученные при выполнении умножения (сек.).

На основе полученных данных, построим таблицу соотношения размеров векторов и времени их умножения:

|  |  |
| --- | --- |
| Количество элементов в матрице | Время выполнения (сек.) |
| 1000000 | 0.004 |
| 2000000 | 0.006 |
| 3000000 | 0.012 |

Таблица 1: Результаты замеров времени умножения векторов.

Таким образом мы можем видеть, что наши теоретические предположения об асимптотике алгоритмов оправданы – действительно время работы эквивалентно асимптотической функции, которую мы посчитали.

# 6. Заключение

По итогу проведения данной работы, нам удалось реализовать удобные объекты для работы с матрицами и векторами. Также по проведенным экспериментам можно сделать вывод, что с некоторыми погрешностями, затраченное на реализацию матричных, векторных операций время, совпадает с ожидаемым относительно асимптотической сложности.

# 7. Литература

1. Павловская Т.А. C/C++, Программирование на языке высокого уровня, 2003.
2. Официальный сайт Stackoverflow. — Режим доступа <https://ru.stackoverflow.com>.
3. Официальный сайт LearnC. — Режим доступа <https://learnc.info/c/>

# 8. Приложение

# umatrix.h

#ifndef \_\_TMATRIX\_H\_\_

#define \_\_TMATRIX\_H\_\_

#include <iostream>

#include <exception>

using namespace std;

const int MAX\_VECTOR\_SIZE = 100000000;

const int MAX\_MATRIX\_SIZE = 10000;

// Шаблон вектора

template <class ValType>

class TVector

{

protected:

ValType \*pVector;

int Size; // размер вектора

int StartIndex; // индекс первого элемента вектора

public:

TVector(int s = 10, int si = 0); // одновременно конструктор по умолчанию

// и конструктор с параметрами

TVector(const TVector &v);

TVector(TVector &&v);// конструктор копирования

virtual ~TVector();

int GetSize() const { return Size; } // размер вектора

int GetStartIndex()const{ return StartIndex; }

void SetSize(int x)

{

Size = x;

}

void SetStartIndex(int x)

{

StartIndex = x;

}

ValType\* GetVector()const { return pVector; }// индекс первого элемента

ValType& operator[](int pos);

ValType& operator[](int pos) const;// доступ

bool operator==(const TVector &v) const; // сравнение

bool operator!=(const TVector &v) const; // сравнение

TVector& operator=(const TVector &v); // присваивание

// скалярные операции

TVector operator+(const ValType &val); // прибавить скаляр

TVector operator-(const ValType &val); // вычесть скаляр

TVector operator\*(const ValType &val); // умножить на скаляр

// векторные операции

TVector operator+(const TVector &v); // сложение

TVector operator-(const TVector &v); // вычитание

ValType operator\*(const TVector &v); // скалярное произведение

int FindIndex()

{

for (int i = (\*this).GetStartIndex(); i < (\*this).GetSize(); i++)

{

if (((\*this)[i] != 0))return i;

}

return -1;

}

// ввод-вывод

friend istream& operator>>(istream &in, TVector &v)

{

for (int i = 0; i < v.Size-v.StartIndex; i++)

in >> v.pVector[i];

return in;

}

friend ostream& operator<<(ostream &out, const TVector &v)

{

for (int i = 0; i < v.Size - v.StartIndex; i++) {

out << v.pVector[i] << ' ';

}

out << endl;

return out;

}

};

template <class ValType>

TVector<ValType>::TVector(int s, int si)

{

if ((s < 0) || (si<0) || (s>MAX\_VECTOR\_SIZE)) throw logic\_error("");

pVector = new ValType[s];

for (int i = 0; i < s; i++)

{

pVector[i] = 0;

}

StartIndex = si;

Size = s + StartIndex;

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

template <class ValType> //конструктор копирования

TVector<ValType>::TVector(const TVector<ValType> &v)

{

pVector = new ValType[v.Size-v.StartIndex];

for (int i = 0; i < v.Size-v.StartIndex; i++)

{

pVector[i] = v.pVector[i];

}

StartIndex = v.StartIndex;

Size = v.Size;

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

template <class ValType> //конструктор копирования

TVector<ValType>::TVector( TVector<ValType> &&v)

{

pVector =v.pVector;

v.pVector = nullptr;

StartIndex = v.StartIndex;

Size = v.Size;

}

template <class ValType>

TVector<ValType>::~TVector()

{

if (pVector != nullptr) {

delete[]pVector;

}

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

template <class ValType> // доступ

ValType& TVector<ValType>::operator[](int pos)

{

if ((pos < StartIndex) || (pos < 0) || (pos >Size)) throw logic\_error("");

return pVector[pos-StartIndex];

}

template <class ValType> // доступ

ValType& TVector<ValType>::operator[](int pos)const

{

if ((pos < StartIndex) || (pos < 0) || (pos > Size)) throw logic\_error("");

return pVector[pos - StartIndex];

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

template <class ValType> // сравнение

bool TVector<ValType>::operator==(const TVector &v) const

{

if (this == &v)return 1;

if ((this->GetSize() == v.GetSize()) && (this->GetStartIndex() == v.GetStartIndex()))

{

for (int i = 0; i < v.GetSize() - v.GetStartIndex(); i++)

{

if (this->pVector[i] != v.pVector[i])

{

return 0;

}

return 1;

}

}

return 0;

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

template <class ValType> // сравнение

bool TVector<ValType>::operator!=(const TVector &v) const

{

return !(\*this==v);

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

template <class ValType> // присваивание

TVector<ValType>& TVector<ValType>::operator=(const TVector &v)

{

if (this == &v)

{

return (\*this);

}

if (this->GetSize() == v.GetSize())

{

for (int i = 0; i < v.Size - v.StartIndex; i++)

{

pVector[i] = v.pVector[i];

}

Size = v.Size;

StartIndex = v.StartIndex;

return (\*this);

}

delete []pVector;

pVector = new ValType[v.Size - v.StartIndex];

for (int i = 0; i < v.Size - v.StartIndex; i++)

{

pVector[i] = v.pVector[i];

}

Size = v.Size;

StartIndex = v.StartIndex;

return (\*this);

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

template <class ValType> // прибавить скаляр

TVector<ValType> TVector<ValType>::operator+(const ValType &val)

{

TVector<ValType>res(\*this);

for (int i = 0; i < res.Size - res.StartIndex; i++)

{

res.pVector[i] += val;

}

return res;

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

template <class ValType> // вычесть скаляр

TVector<ValType> TVector<ValType>::operator-(const ValType &val)

{

TVector<ValType>res(\*this);

for (int i = 0; i < res.Size - res.StartIndex; i++)

{

res.pVector[i] -= val;

}

return res;

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

template <class ValType> // умножить на скаляр

TVector<ValType> TVector<ValType>::operator\*(const ValType &val)

{

TVector<ValType>res(\*this);

for (int i = 0; i < res.Size - res.StartIndex; i++)

{

res.pVector[i] \*= val;

}

return res;

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

template <class ValType> // сложение

TVector<ValType> TVector<ValType>::operator+(const TVector<ValType> &v)

{

if (this->GetSize() != v.GetSize()) throw logic\_error("");

TVector<ValType>res(\*this);

for (int i = 0; i < res.Size - res.StartIndex; i++)

{

res.pVector[i] =res.pVector[i]+ v.pVector[i];

}

return res;

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

template <class ValType> // вычитание

TVector<ValType> TVector<ValType>::operator-(const TVector<ValType> &v)

{

if (this->GetSize() != v.GetSize()) throw logic\_error("");

TVector<ValType>res(\*this);

for (int i = 0; i < res.Size - res.StartIndex; i++)

{

res.pVector[i] = res.pVector[i] - v.pVector[i];

}

return res;

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

template <class ValType> // скалярное произведение

ValType TVector<ValType>::operator\*(const TVector<ValType> &v)

{

if (this->GetSize() != v.GetSize()) throw logic\_error("");

float res=0;

for (int i = 0; i < v.Size - v.StartIndex; i++)

{

res = res + v.pVector[i]\*this->pVector[i];

}

return sqrt(res);

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

// Верхнетреугольная матрица

template <class ValType>

class TMatrix : public TVector<TVector<ValType> >

{

public:

TMatrix(int s = 10);

TMatrix(const TMatrix &mt); // копирование

TMatrix(const TVector<TVector<ValType> > &mt); // преобразование типа

bool operator==(const TMatrix &mt) const; // сравнение

bool operator!=(const TMatrix &mt) const; // сравнение

TMatrix& operator= (const TMatrix &mt); // присваивание

TMatrix operator+ (const TMatrix &mt); // сложение

TMatrix operator- (const TMatrix &mt); // вычитание

TMatrix( TVector<TVector<ValType> >&& mt);

// ввод / вывод

friend istream& operator>>(istream &in, TMatrix &mt)

{

for (int i = 0; i < mt.Size; i++)

in >> mt.pVector[i];

return in;

}

friend ostream & operator<<( ostream &out, const TMatrix &mt)

{

for (int i = 0; i < mt.Size; i++)

out << mt.pVector[i] << endl;

return out;

}

};

template <class ValType>

TMatrix<ValType>::TMatrix(int s): TVector<TVector<ValType> >(s)

{

if (s > MAX\_MATRIX\_SIZE) throw logic\_error("");

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

{

TVector<ValType>res(s-i, i);

this->pVector[i] = res;

}

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

template <class ValType> // конструктор копирования

TMatrix<ValType>::TMatrix(const TMatrix<ValType> &mt):

TVector<TVector<ValType> >(mt)

{

for (int i = this->StartIndex; i < this->Size; i++)

{

TVector<ValType>res(mt[i]);

(\*this)[i] = res;

}

}

template <class ValType> // конструктор преобразования типа

TMatrix<ValType>::TMatrix(const TVector<TVector<ValType> > &mt):

TVector<TVector<ValType> >(mt)

{

for (int i = (\*this).GetStartIndex(); i < (\*this).GetSize(); i++)

{

TVector<ValType>ms(mt[i]);

(\*this)[i] = ms;

}

}

template <class ValType> // сравнение

bool TMatrix<ValType>::operator==(const TMatrix<ValType> &mt) const

{

if (this == &mt) return 1;

if (((\*this).GetStartIndex() == mt.GetStartIndex()) && ((\*this).GetSize() == mt.GetSize()))

{

for (int i = mt.GetStartIndex(); i < mt.GetSize(); i++)

{

if ((\*this)[i] != mt[i])

{

return false;

}

}

return true;

}

return false;

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

template <class ValType> // сравнение

bool TMatrix<ValType>::operator!=(const TMatrix<ValType> &mt) const

{

return !((\*this) == mt);

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

template <class ValType> // присваивание

TMatrix<ValType>& TMatrix<ValType>::operator=(const TMatrix<ValType> &mt)

{

if (&mt == this)

{

return (\*this);

}

if (this->GetSize() == mt.GetSize())

{

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

{

(\*this).pVector[i] = mt.pVector[i];

}

return(\*this);

}

delete[]pVector;

pVector = new TVector<ValType>[mt.Size-mt.StartIndex];

(\*this).Size = mt.Size;

(\*this).StartIndex = mt.StartIndex;

for (int i = (\*this).GetStartIndex(); i < (\*this).GetSize(); i++)

{

(\*this)[i] = mt[i];

}

return (\*this);

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

template <class ValType> // сложение

TMatrix<ValType> TMatrix<ValType>::operator+(const TMatrix<ValType> &mt)

{

// благодаря наследованию от TVector<TVector<ValType> > operator+

// уже есть, надо только его вызвать

if (mt.Size != (\*this).GetSize()) throw logic\_error("incorrect size");

TMatrix<ValType>Res(\*this);

for (int i = Res.GetStartIndex(); i < Res.GetSize(); i++)

{

Res[i] = (\*this)[i] + mt[i];

}

return Res;

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

template <class ValType> // вычитание

TMatrix<ValType> TMatrix<ValType>::operator-(const TMatrix<ValType> &mt)

{

if (mt.GetSize() != (\*this).GetSize()) throw logic\_error("incorrect size");

TMatrix<ValType>Res(\*this);

for (int i = Res.GetStartIndex(); i < Res.GetSize(); i++)

{

Res[i] = (\*this)[i] - mt[i];

}

return Res;

} /\*--------------------------------------------------s-----------------------\*/

template <class ValType> // вычитание

TMatrix<ValType>::TMatrix( TVector<TVector<ValType> > &&mt):TVector<TVector<ValType> >(move(mt))

{

bool f = true;

for (int i = 0; i < (\*this).GetSize(); i++)

{

if ((\*this)[i][i] == 0)

{

int j = -1;

int k = i+1;

while ((j == -1) && (k < (\*this).GetSize()))

{

if((\*this)[k][i]!=0)j = k;

k++;

}

if (j != -1)

{

swap((\*this)[i], (\*this)[j]);

f = true;

}

else

{

f = false;

}

}

if (f) {

ValType x = 1 / (\*this)[i][i];

(\*this)[i] = (\*this)[i] \* x;

for (int j = i + 1; j < mt.GetSize(); j++)

{

(\*this)[j] = (\*this)[j] - ((\*this)[i] \* (\*this)[j][i]);

}

}

}

cout << (\*this);

for (int i = (\*this).GetStartIndex(); i < (\*this).GetSize(); i++)

{

for (int j = i; j < (\*this).GetSize(); j++)

{

(\*this)[i][j - i] = move((\*this)[i][j]);

}

(\*this)[i].SetStartIndex(i);

}

}

// TVector О3 Л2 П4 С6

// TMatrix О2 Л2 П3 С3

#endif

# sample\_matrix.cpp

#include <iostream>

#include "utmatrix.h"

#include <iomanip>

//---------------------------------------------------------------------------

int main()

{

TMatrix<int> a(2000), b(2000);

TMatrix<int>a1(4000), b1(4000);

TMatrix<int>a2(8000), b2(8000);

int i, j;

setlocale(LC\_ALL, "Russian");

cout << "Замер реальной времени работы операций"

<< endl;

time\_t s1, s2, s3, e1, e2, e3;

for (i = 0; i < 2000; i++)

for (j = i; j < 2000; j++ )

{

a[i][j] = i \* 10 + j;

b[i][j] = (i \* 10 + j) \* 100;

}

for (i = 0; i < 4000; i++)

for (j = i; j < 4000; j++)

{

a1[i][j] = i \* 10 + j;

b1[i][j] = (i \* 10 + j) \* 100;

}

for (i = 0; i < 8000; i++)

for (j = i; j < 8000; j++)

{

a2[i][j] = i \* 10 + j;

b2[i][j] = (i \* 10 + j) \* 100;

}

s1 = clock();

a + b;

e1 = clock();

s2 = clock();

a1 + b1;

e2 = clock();

s3 = clock();

a2 + b2;

e3 = clock();

cout << (double)(e1 - s1) / CLOCKS\_PER\_SEC <<

endl <<(double)(e2 - s2) / CLOCKS\_PER\_SEC << endl << (double)(e3 - s3) / CLOCKS\_PER\_SEC << endl;

time\_t s4, e4, s5, e5, s6, e6;

TVector<int>A1(1000000), B1(1000000),A2(2000000),B2(2000000),A3(3000000),B3(3000000);

for (int i = 0; i < 1000000; i++)

{

A1[i] = i;

B1[i] = i;

}

for (int i = 0; i < 2000000; i++)

{

A2[i] = i;

B2[i] = i;

}

for (int i = 0; i < 3000000; i++)

{

A3[i] = i;

B3[i] = i;

}

s4 = clock();

A1\*B1;

e4 = clock();

s5 = clock();

A2\*B2;

e5 = clock();

s6 = clock();

A3\*B3;

e6 = clock();

cout << (double)(e4 - s4) / CLOCKS\_PER\_SEC << endl << (double)(e5 - s5) / CLOCKS\_PER\_SEC <<

endl << (double)(e6 - s6) / CLOCKS\_PER\_SEC << endl;

return 0;

}

# test\_tmatrix.cpp

#include "utmatrix.h"

#include <gtest.h>

TEST(TMatrix, can\_create\_matrix\_with\_positive\_length)

{

ASSERT\_NO\_THROW(TMatrix<int> m(5));

}

TEST(TMatrix, cant\_create\_too\_large\_matrix)

{

ASSERT\_ANY\_THROW(TMatrix<int> m(MAX\_MATRIX\_SIZE + 1));

}

TEST(TMatrix, throws\_when\_create\_matrix\_with\_negative\_length)

{

ASSERT\_ANY\_THROW(TMatrix<int> m(-5));

}

TEST(TMatrix, can\_create\_copied\_matrix)

{

TMatrix<int> m(5);

ASSERT\_NO\_THROW(TMatrix<int> m1(m));

}

TEST(TMatrix, copied\_matrix\_is\_equal\_to\_source\_one)

{

TMatrix<int> A1(5);

TMatrix<int> A2(A1);

EXPECT\_EQ(A1, A2);

}

TEST(TMatrix, copied\_matrix\_has\_its\_own\_memory)

{

TMatrix<int> A1(5);

TMatrix<int> A2(A1);

EXPECT\_NE(A1.GetVector(), A2.GetVector());

}

TEST(TMatrix, can\_get\_size)

{

TMatrix<int>A(8);

EXPECT\_EQ(A.GetSize(), 8);

}

TEST(TMatrix, can\_set\_and\_get\_element)

{

TMatrix<int>A(8);

A[1][2] = 9;

EXPECT\_EQ(A[1][2], 9);

}

TEST(TMatrix, throws\_when\_set\_element\_with\_negative\_index)

{

TMatrix<int>A(3);

ASSERT\_ANY\_THROW(A[-1][-2]);

}

TEST(TMatrix, throws\_when\_set\_element\_with\_too\_large\_index)

{

TMatrix<int>A(3);

ASSERT\_ANY\_THROW(A[5][5]);

}

TEST(TMatrix, can\_assign\_matrix\_to\_itself)

{

TMatrix<int>A(3);

ASSERT\_NO\_THROW(A = A);

}

TEST(TMatrix, can\_assign\_matrices\_of\_equal\_size)

{

TMatrix<int>A1(3);

TMatrix<int>A2(3);

auto i = A2.GetVector();

A2 = A1;

EXPECT\_EQ(i, A2.GetVector());

}

TEST(TMatrix, assign\_operator\_change\_matrix\_size)

{

TMatrix<int>A1(3);

TMatrix<int>A2(4);

auto i = A2.GetVector();

A2 = A1;

EXPECT\_EQ(A2.GetSize(), 3);

}

TEST(TMatrix, can\_assign\_matrices\_of\_different\_size)

{

TMatrix<int>A1(3);

TMatrix<int>A2(4);

ASSERT\_NO\_THROW(A2 = A1);

}

TEST(TMatrix, compare\_equal\_matrices\_return\_true)

{

TMatrix<int>A1(10);

TMatrix<int>A2(10);

EXPECT\_EQ(A1, A2);

}

TEST(TMatrix, compare\_matrix\_with\_itself\_return\_true)

{

TMatrix<int>A1(10);

EXPECT\_EQ(A1, A1);

}

TEST(TMatrix, matrices\_with\_different\_size\_are\_not\_equal)

{

TMatrix<int>A1(10);

TMatrix<int>A2(11);

EXPECT\_NE(A1, A2);

}

TEST(TMatrix, can\_add\_matrices\_with\_equal\_size)

{

TMatrix<int>A1(10);

TMatrix<int>A2(10);

ASSERT\_NO\_THROW(A1 + A2);

}

TEST(TMatrix, cant\_add\_matrices\_with\_not\_equal\_size)

{

TMatrix<int>A1(10);

TMatrix<int>A2(12);

ASSERT\_ANY\_THROW(A1 + A2);

}

TEST(TMatrix, can\_subtract\_matrices\_with\_equal\_size)

{

TMatrix<int>A1(10);

TMatrix<int>A2(10);

ASSERT\_NO\_THROW(A1 - A2);

}

TEST(TMatrix, cant\_subtract\_matrixes\_with\_not\_equal\_size)

{

TMatrix<int>A1(10);

TMatrix<int>A2(12);

ASSERT\_ANY\_THROW(A1 - A2);

}

# test\_tvector.cpp

#include "utmatrix.h"

#include <gtest.h>

TEST(TVector, can\_create\_vector\_with\_positive\_length)

{

ASSERT\_NO\_THROW(TVector<int> v(5));

}

TEST(TVector, cant\_create\_too\_large\_vector)

{

ASSERT\_ANY\_THROW(TVector<int> v(MAX\_VECTOR\_SIZE + 1));

}

TEST(TVector, can\_create\_vector\_with\_max\_size)

{

ASSERT\_NO\_THROW(TVector<int> v(MAX\_VECTOR\_SIZE));

}

TEST(TVector, can\_create\_vector\_with\_zero\_size)

{

ASSERT\_NO\_THROW(TVector<int> v(0));

}

TEST(TVector, throws\_when\_create\_vector\_with\_negative\_length)

{

ASSERT\_ANY\_THROW(TVector<int> v(-5));

}

TEST(TVector, throws\_when\_create\_vector\_with\_negative\_startindex)

{

ASSERT\_ANY\_THROW(TVector<int> v(5, -2));

}

TEST(TVector, can\_create\_copied\_vector)

{

TVector<int> v(10);

ASSERT\_NO\_THROW(TVector<int> v1(v));

}

TEST(TVector, copied\_vector\_is\_equal\_to\_source\_one)

{

TVector<int>A1(5);

TVector<int>A2(A1);

EXPECT\_EQ(A1, A2);

}

TEST(TVector, copied\_vector\_has\_its\_own\_memory)

{

TVector<int>A1(5);

TVector<int>A2(A1);

EXPECT\_NE(A1.GetVector(), A2.GetVector());

}

TEST(TVector, can\_get\_size)

{

TVector<int> v(4);

EXPECT\_EQ(4, v.GetSize());

}

TEST(TVector, can\_get\_start\_index)

{

TVector<int> v(4, 2);

EXPECT\_EQ(2, v.GetStartIndex());

}

TEST(TVector, can\_set\_and\_get\_element)

{

TVector<int> v(4);

v[0] = 4;

EXPECT\_EQ(4, v[0]);

}

TEST(TVector, throws\_when\_set\_element\_with\_negative\_index)

{

TVector<int> v(4);

ASSERT\_ANY\_THROW(v[-1]);

}

TEST(TVector, throws\_when\_set\_element\_with\_too\_large\_index)

{

TVector<int> v(4);

ASSERT\_ANY\_THROW(v[6]);

}

TEST(TVector, can\_assign\_vector\_to\_itself)

{

TVector<int> v(4);

ASSERT\_NO\_THROW(v = v);

}

TEST(TVector, can\_assign\_vectors\_of\_equal\_size)

{

TVector<int> v(4);

TVector<int>v1(4);

auto i = v1.GetVector();

v1 = v;

EXPECT\_EQ(i, v1.GetVector());

}

TEST(TVector, assign\_operator\_change\_vector\_size)

{

TVector<int> v(4);

TVector<int>v1(6);

v1 = v;

EXPECT\_EQ(v1.GetSize(), 4);

}

TEST(TVector, can\_assign\_vectors\_of\_different\_size)

{

TVector<int> v(4);

TVector<int>v1(6);

ASSERT\_NO\_THROW(v1 = v);

}

TEST(TVector, compare\_equal\_vectors\_return\_true)

{

TVector<int> v(4);

TVector<int>v1(4);

EXPECT\_EQ(v1 ==v,1);

}

TEST(TVector, compare\_vector\_with\_itself\_return\_true)

{

TVector<int> v(4);

EXPECT\_EQ(v == v, 1);

}

TEST(TVector, vectors\_with\_different\_size\_are\_not\_equal)

{

TVector<int> v(4);

TVector<int>v1(5);

EXPECT\_EQ(v1 == v, 0);

}

TEST(TVector, can\_add\_scalar\_to\_vector)

{

TVector<int> v(4);

TVector<int> v1(4);

v = v + 4;

for (int i = v1.GetStartIndex(); i < v1.GetSize(); i++)

{

v1[i] = 4;

}

EXPECT\_EQ(v1, v);

}

TEST(TVector, can\_subtract\_scalar\_from\_vector)

{

TVector<int> v(4);

TVector<int> v1(4);

v = v - 4;

for (int i = v1.GetStartIndex(); i < v1.GetSize(); i++)

{

v1[i] = -4;

}

EXPECT\_EQ(v1, v);

}

TEST(TVector, can\_multiply\_scalar\_by\_vector)

{

TVector<int> v(4);

TVector<int> v1(4);

v = v + 1;

v = v \* 3;

for (int i = v1.GetStartIndex(); i < v1.GetSize(); i++)

{

v1[i] = 3;

}

EXPECT\_EQ(v1, v);

}

TEST(TVector, can\_add\_vectors\_with\_equal\_size)

{

TVector<int> v(4);

TVector<int> v1(4);

TVector<int> v3(4);

v = v + 4;

v1 = v1 + 3;

v3 = v3 + 7;

EXPECT\_EQ(v1+v, v3);

}

TEST(TVector, cant\_add\_vectors\_with\_not\_equal\_size)

{

TVector<int> v(4);

TVector<int> v1(2);

ASSERT\_ANY\_THROW(v + v1);

}

TEST(TVector, can\_subtract\_vectors\_with\_equal\_size)

{

TVector<int> v(4);

TVector<int> v1(4);

ASSERT\_NO\_THROW(v + v1);

}

TEST(TVector, cant\_subtract\_vectors\_with\_not\_equal\_size)

{

TVector<int> v(4);

TVector<int> v1(2);

ASSERT\_ANY\_THROW(v - v1);

}

TEST(TVector, can\_multiply\_vectors\_with\_equal\_size)

{

TVector<int> v(4);

TVector<int> v1(4);

ASSERT\_NO\_THROW(v \* v1);

}

TEST(TVector, cant\_multiply\_vectors\_with\_not\_equal\_size)

{

TVector<int> v(4);

TVector<int> v1(2);

ASSERT\_ANY\_THROW(v \* v1);

}