Федеральное государственное автономное образовательное учреждение высшего образования

**«Национальный исследовательский Нижегородский государственный университет**

**им. Н.И. Лобачевского»**

**Институт информационных технологий, математики и механики**

Направление подготовки: Прикладная математика и информатика

Отчёт по лабораторной работе

**Матрицы и их виды**

**Выполнил:** студент группы 3821Б1ПМ3

Благодеров В.А.

**Проверил:** заведующий лабораторией

суперкомпьютерных технологий и

высокопроизводительных вычислений

Лебедев И.Г

Нижний Новгород

2022 г.

Содержание

Введение 3

1. Постановка задачи 4

2. Руководство пользователя 5

3. Руководство программиста 6

a. Описание структуры программы 6

b. Описание структуры данных 6

c. Описание алгоритмов 7

4. Эксперименты 12

Заключение 18

Литература 19

Приложения 20

Приложение 1 20

Приложение 2 24

Приложение 3 27

Приложение 4 30

Приложение 5 32

Введение

Матрица – математический объект, записываемый в виде прямоугольной таблицы элементов, который представляет собой совокупность строк и столбцов, на пересечении которых находятся его элементы. Количество строк и столбцов задаёт размер матрицы. Матрицы широко применяются в математике для компактной записи систем линейных алгебраических или дифференциальных уравнений. В этом случае количество строк матрицы соответствует числу уравнений, а количество столбцов – количеству неизвестных. В результате над матрицей можно производить действия по правилам матричной алгебры. Определение матрицы возможно выразить через понятие вектор. Вектор – набор чисел, состоящий из n элементов. Тогда матрица из m строк и n столбцов может быть определена как вектор из n элементов, где каждый элемент является векторов из m элементов.

1. Постановка задачи

Ставится задача создания программных средств, поддерживающих эффективное хранение матриц специального вида и выполнение основных операций над ними: сложение/вычитание; умножение; копирование; сравнение.

Программные средства должны содержать:

Класс Вектор (на шаблонах);

Класс Матрица (на шаблонах);

Тестирование.

1. Руководство пользователя

Чтобы воспользоваться программой необходимо создать объекты из класса «TDynamicVector» или «TSquareDynamicMatrix» или «TTriangleDynamicMatrix». Для создания объекта применяются конструкторы.

|  |
| --- |
| TDynamicVector(); // Конструктор по умолчанию для вектора  TDynamicVector(int n); // Конструктор для вектора длины n  TSquareDynamicMatrix(); // Конструктор по умолчанию для квадратной матрицы  TTriangleDynamicMatrix(); // Конструктор по умолчанию для треугольной матрицы  TSquareDynamicMatrix(int n); // Конструктор для квадратной матрицы n\*n  TTriangleDynamicMatrix(int n); // Конструктор для квадратной треугольной матрицы n\*n |

Фрагмент кода 1. Конструкторы

|  |
| --- |
| TDynamicVector<int> A(5); // Создали вектор длины 5  TSquareDynamicMatrix<int> m1(2); // Создали матрицу длины 2 |

Фрагмент кода 2. Пример создания объектов

После создания объекта можно воспользоваться функционалом классов.

|  |
| --- |
| TDynamicVector<int> A(5); // Создали вектор A  TDynamicVector<int> C(5); // Создали вектор C  TDynamicVector<int> D(5); // Создали вектор D  C = A + D; // В вектор C записали сумму векторов  std::cout << C << std::endl; // Вывели вектор C на экран |

Фрагмент кода 3. Пример пользования программой

1. Руководство программиста
   1. Описание структуры программы

Представляется следующая модульная структура программы:

1. TVector.h, TVector.cpp – модуль, реализующий структуру данных Вектор.
2. TTriangleMatrix.h, TSquareMatrix.h, TTriangleMartix.cpp, TSquareMatrix.cpp – модуль, реализующий структуру данных Матрица.
3. sample\_matrix.cpp, test\_tvector.cpp, test\_tmatrix.cpp, test\_main.cpp – модуль, реализующий тестирование.
   1. Описание структуры данных

Используются шаблоны. Класс Вектор:

|  |
| --- |
| protected:  int len; // Длина вектора  T\* pMem; // Массив элементов вектора  public:  TDynamicVector(); // Конструктор по умолчанию  TDynamicVector(int n); // Конструктор инициализатор  TDynamicVector(T\* arr, int s); // Конструктор копирования  TDynamicVector(const TDynamicVector<T>& v); // Конструктор копирования  TDynamicVector(TDynamicVector<T>&& v) noexcept; // Конструктор перемещения  ~TDynamicVector(); // Деструктор  int size(); // Получение размера  void Resize(int nlen); // Изменение размера  T& operator[](int i); // Индексация  const T& operator[](int i) const; // Индексация  bool operator==(const TDynamicVector<T>& v); // Равенство  bool operator!=(const TDynamicVector<T>& v); // Не равенство  TDynamicVector<T> operator+(T n); // Сложение с константой  TDynamicVector<T> operator-(double n); // Вычитание с константой  TDynamicVector<T> operator\*(double n); // Умножение с константой  TDynamicVector<T> operator+(const TDynamicVector<T>& v); // Сложение векторов  TDynamicVector<T> operator-(const TDynamicVector<T>& v); // Вычитание векторов  TDynamicVector<T> operator\*(const TDynamicVector<T>& v); // Умножение векторов  TDynamicVector<T>& operator=(const TDynamicVector<T>& v); // Присваивание векторов  TDynamicVector<T>& operator=(TDynamicVector<T>&& v) noexcept; // Перемещающее присваивание  friend istream& operator>>(istream& istr, const TDynamicVector<T>& v); // Ввод вектора  friend ostream& operator<<(ostream& ostr, const TDynamicVector<T>& v); // Вывод вектора |

Фрагмент кода 4. Класс «TDynamicVector».

Класс Матрица:

|  |
| --- |
| TSquareDynamicMatrix(); // Конструктор по умолчанию  TSquareDynamicMatrix(int n); // Конструктор инициализатор  TSquareDynamicMatrix(const TSquareDynamicMatrix<T>& m); // Конструктор копирования  bool operator==(const TSquareDynamicMatrix<T>& m); // Равенство  bool operator!=(const TSquareDynamicMatrix<T>& m); // Не равенство  TSquareDynamicMatrix<T> operator\*(const T& v); // Умножение матрицы на константу  TDynamicVector<T> operator\*(TDynamicVector<T>& v); // Умножение матрицы на вектор  TSquareDynamicMatrix<T> operator+(const TSquareDynamicMatrix<T>& m); // Сложение матриц  TSquareDynamicMatrix<T> operator-(const TSquareDynamicMatrix<T>& m); // Вычитание матриц  TSquareDynamicMatrix<T> operator\*(const TSquareDynamicMatrix<T>& m); // Умножение матриц  TSquareDynamicMatrix<T> operator=(const TSquareDynamicMatrix<T>& m); // Присваивание матриц  friend istream& operator>>(istream& istr, const TSquareDynamicMatrix<T>& m); // Ввод матриц  friend ostream& operator<<(ostream& ostr, const TSquareDynamicMatrix<T>& m); // Вывод матриц |

Фрагмент кода 5. Класс «TSquareDynamicMatrix»

В классе «TTriangleDynamicMatrix» методы аналогичные классу «TSquareDynamicMatrix».

* 1. Описание алгоритмов

Программа реализует различные операции над векторами и матрицами. Рассмотрим некоторые из них.

1. Сложение векторов (operator +). Суть метода. Функция принимает вектор, далее выполняется тело функции. Возвращается новый вектор (Смотри блок-схему 1).

TDynamicVector<T> res(\*this);

((len == v.len) && (v.pMem != 0) && (pMem != 0))

i=0, len

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

res

Да

Нет

Блок-схема 1. «Сложение векторов»

1. Присваивание векторов (operator =). Суть метода. Функция принимает вектор, далее выполняется тело функции. Возвращается ссылка на вектор, а не сам вектор (Смотри блок-схему 2).

(this == &v)

(v.pMem == 0)

delete[] pMem;

pMem = 0;

len = 0;

(pMem != 0)

len = v.len;

pMem = new T[len];

i=0, len

pMem[i] = v.pMem[i];

\*this

\*this

\*this

Да

Нет

Да

Нет

Да

Блок-схема 2. «Присваивание векторов»

1. Умножение матриц (operator \*). Суть метода. Функция принимает матрицу, далее выполняется тело функции. Возвращается матрица (Смотри блок-схему 3).

(this->len == v.len)

TSquareDynamicMatrix<T> res(\*this);

i=0, len

res[i][j] += this->pMem[i][k] \* m.pMem[k][j];

res

j=0, len

k=0, len

Да

res[i][j] = 0;

Блок-схема 3. «Умножение матриц»

1. Равенство матриц (operator ==). Суть метода. Принимает матрицу, далее выполняется тело функции. Возвращается значение 0 или 1, где 0 – ложь, а 1 – истина (Смотри блок-схему 4).

Блок-схема 4. «Оператор == для матриц»

((this->len != v.len)))

i=0, len

(this->pMem[i][j] != m.pMem[i][j])

false

false

true

j=0, len

Да

Нет

Да

Нет

1. Эксперименты

Проведём несколько экспериментов и проверим работоспособность. Создадим пару объектов и выполним над нами операции.

|  |
| --- |
| TTriangleDynamicMatrix<int> a(5), b(5), c(5);  TSquareDynamicMatrix<int> f(5), d(5), v(5);  TDynamicVector<int> k(5), r(5);  setlocale(LC\_ALL, "Russian");  std::cout << "Тестирование класс работы с матрицами" << std::endl;  for (int i = 0; i < 5; i++)  for (int j = 0; j <= i; j++)  {  a[i][j] = i \* 10 + j;  b[i][j] = (i \* 10 + j) \* 100;  }  c = a + b;  cout << "Matrix a = " << endl << a << endl;  cout << "Matrix b = " << endl << b << endl;  cout << "Matrix c = a + b" << endl << c << endl;  for (int i = 0; i < 5; i++)  for (int j = 0; j < 5; j++)  {  f[i][j] = i + 1;  d[i][j] = i + 1;  }  v = f \* d;  r = v \* k;  cout << "Matrix f = " << endl << f << endl;  cout << "Matrix d = " << endl << d << endl;  cout << "Matrix v = f + d" << endl << v << endl;  cout << "Vector k = " << endl << k << endl;  cout << "Vector r = v \* k" << endl << r << endl; |

Фрагмент кода 6. Создание

Получившийся результат:



Рис. 1. Работа с треугольными матрицами.

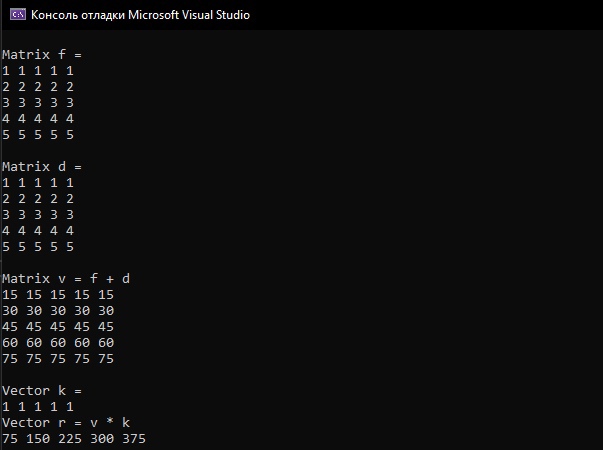


Рис. 2. Работа с квадратными матрицами.

Протестируем работоспособность с помощью тестов.

|  |
| --- |
| [==========] Running 54 tests from 3 test cases.  [----------] Global test environment set-up.  [----------] 17 tests from TTriangleDynamicMatrix  [ RUN ] TTriangleDynamicMatrix.can\_create\_matrix\_with\_positive\_length  [ OK ] TTriangleDynamicMatrix.can\_create\_matrix\_with\_positive\_length (0 ms)  [ RUN ] TTriangleDynamicMatrix.throws\_when\_create\_matrix\_with\_negative\_length  [ OK ] TTriangleDynamicMatrix.throws\_when\_create\_matrix\_with\_negative\_length (0 ms)  [ RUN ] TTriangleDynamicMatrix.can\_create\_copied\_matrix  [ OK ] TTriangleDynamicMatrix.can\_create\_copied\_matrix (0 ms)  [ RUN ] TTriangleDynamicMatrix.can\_get\_size  [ OK ] TTriangleDynamicMatrix.can\_get\_size (0 ms)  [ RUN ] TTriangleDynamicMatrix.compare\_equal\_matrices\_return\_true  [ OK ] TTriangleDynamicMatrix.compare\_equal\_matrices\_return\_true (0 ms)  [ RUN ] TTriangleDynamicMatrix.compare\_matrix\_with\_itself\_return\_true  [ OK ] TTriangleDynamicMatrix.compare\_matrix\_with\_itself\_return\_true (0 ms)  [ RUN ] TTriangleDynamicMatrix.matrices\_with\_different\_size\_are\_not\_equal  [ OK ] TTriangleDynamicMatrix.matrices\_with\_different\_size\_are\_not\_equal (0 ms)  [ RUN ] TTriangleDynamicMatrix.cant\_add\_matrices\_with\_not\_equal\_size  [ OK ] TTriangleDynamicMatrix.cant\_add\_matrices\_with\_not\_equal\_size (0 ms)  [ RUN ] TTriangleDynamicMatrix.can\_add\_matrices\_with\_equal\_size  [ OK ] TTriangleDynamicMatrix.can\_add\_matrices\_with\_equal\_size (0 ms)  [ RUN ] TTriangleDynamicMatrix.cant\_subtract\_matrixes\_with\_not\_equal\_size  [ OK ] TTriangleDynamicMatrix.cant\_subtract\_matrixes\_with\_not\_equal\_size (0 ms)  [ RUN ] TTriangleDynamicMatrix.can\_subtract\_matrixes\_with\_equal\_size  [ OK ] TTriangleDynamicMatrix.can\_subtract\_matrixes\_with\_equal\_size (0 ms)  [ RUN ] TTriangleDynamicMatrix.cant\_multiply\_matrixes\_with\_not\_equal\_size  [ OK ] TTriangleDynamicMatrix.cant\_multiply\_matrixes\_with\_not\_equal\_size (0 ms)  [ RUN ] TTriangleDynamicMatrix.can\_equate\_matrixes  [ OK ] TTriangleDynamicMatrix.can\_equate\_matrixes (0 ms)  [ RUN ] TTriangleDynamicMatrix.can\_multiply\_matrixes\_with\_equal\_size  [ OK ] TTriangleDynamicMatrix.can\_multiply\_matrixes\_with\_equal\_size (0 ms)  [ RUN ] TTriangleDynamicMatrix.can\_multiply\_matrix\_by\_vector  [ OK ] TTriangleDynamicMatrix.can\_multiply\_matrix\_by\_vector (0 ms)  [ RUN ] TTriangleDynamicMatrix.cant\_multiply\_matrix\_by\_vector\_with\_not\_equal  [ OK ] TTriangleDynamicMatrix.cant\_multiply\_matrix\_by\_vector\_with\_not\_equal (0 ms)  [ RUN ] TTriangleDynamicMatrix.can\_multiply\_matrix\_by\_scalar  [ OK ] TTriangleDynamicMatrix.can\_multiply\_matrix\_by\_scalar (0 ms)  [----------] 17 tests from TTriangleDynamicMatrix (7 ms total)  [----------] 17 tests from TSquareDynamicMatrix  [ RUN ] TSquareDynamicMatrix.can\_create\_matrix\_with\_positive\_length  [ OK ] TSquareDynamicMatrix.can\_create\_matrix\_with\_positive\_length (0 ms)  [ RUN ] TSquareDynamicMatrix.throws\_when\_create\_matrix\_with\_negative\_length  [ OK ] TSquareDynamicMatrix.throws\_when\_create\_matrix\_with\_negative\_length (0 ms)  [ RUN ] TSquareDynamicMatrix.can\_create\_copied\_matrix  [ OK ] TSquareDynamicMatrix.can\_create\_copied\_matrix (0 ms)  [ RUN ] TSquareDynamicMatrix.can\_get\_size  [ OK ] TSquareDynamicMatrix.can\_get\_size (0 ms)  [ RUN ] TSquareDynamicMatrix.compare\_equal\_matrices\_return\_true  [ OK ] TSquareDynamicMatrix.compare\_equal\_matrices\_return\_true (0 ms)  [ RUN ] TSquareDynamicMatrix.compare\_matrix\_with\_itself\_return\_true  [ OK ] TSquareDynamicMatrix.compare\_matrix\_with\_itself\_return\_true (0 ms)  [ RUN ] TSquareDynamicMatrix.matrices\_with\_different\_size\_are\_not\_equal  [ OK ] TSquareDynamicMatrix.matrices\_with\_different\_size\_are\_not\_equal (0 ms)  [ RUN ] TSquareDynamicMatrix.cant\_add\_matrices\_with\_not\_equal\_size  [ OK ] TSquareDynamicMatrix.cant\_add\_matrices\_with\_not\_equal\_size (0 ms)  [ RUN ] TSquareDynamicMatrix.can\_add\_matrices\_with\_equal\_size  [ OK ] TSquareDynamicMatrix.can\_add\_matrices\_with\_equal\_size (0 ms)  [ RUN ] TSquareDynamicMatrix.cant\_subtract\_matrixes\_with\_not\_equal\_size  [ OK ] TSquareDynamicMatrix.cant\_subtract\_matrixes\_with\_not\_equal\_size (0 ms)  [ RUN ] TSquareDynamicMatrix.can\_subtract\_matrixes\_with\_equal\_size  [ OK ] TSquareDynamicMatrix.can\_subtract\_matrixes\_with\_equal\_size (0 ms)  [ RUN ] TSquareDynamicMatrix.cant\_multiply\_matrixes\_with\_not\_equal\_size  [ OK ] TSquareDynamicMatrix.cant\_multiply\_matrixes\_with\_not\_equal\_size (0 ms)  [ RUN ] TSquareDynamicMatrix.can\_equate\_matrixes  [ OK ] TSquareDynamicMatrix.can\_equate\_matrixes (0 ms)  [ RUN ] TSquareDynamicMatrix.can\_multiply\_matrixes\_with\_equal\_size  [ OK ] TSquareDynamicMatrix.can\_multiply\_matrixes\_with\_equal\_size (0 ms)  [ RUN ] TSquareDynamicMatrix.can\_multiply\_matrix\_by\_vector  [ OK ] TSquareDynamicMatrix.can\_multiply\_matrix\_by\_vector (0 ms)  [ RUN ] TSquareDynamicMatrix.cant\_multiply\_matrix\_by\_vector\_with\_not\_equal  [ OK ] TSquareDynamicMatrix.cant\_multiply\_matrix\_by\_vector\_with\_not\_equal (0 ms)  [ RUN ] TSquareDynamicMatrix.can\_multiply\_matrix\_by\_scalar  [ OK ] TSquareDynamicMatrix.can\_multiply\_matrix\_by\_scalar (0 ms)  [----------] 17 tests from TSquareDynamicMatrix (16 ms total)  [----------] 20 tests from TDynamicVector  [ RUN ] TDynamicVector.can\_create\_vector\_with\_positive\_length  [ OK ] TDynamicVector.can\_create\_vector\_with\_positive\_length (0 ms)  [ RUN ] TDynamicVector.throws\_when\_create\_vector\_with\_negative\_length  [ OK ] TDynamicVector.throws\_when\_create\_vector\_with\_negative\_length (0 ms)  [ RUN ] TDynamicVector.can\_create\_copied\_vector  [ OK ] TDynamicVector.can\_create\_copied\_vector (0 ms)  [ RUN ] TDynamicVector.copied\_vector\_has\_its\_own\_memory  [ OK ] TDynamicVector.copied\_vector\_has\_its\_own\_memory (0 ms)  [ RUN ] TDynamicVector.can\_get\_size  [ OK ] TDynamicVector.can\_get\_size (0 ms)  [ RUN ] TDynamicVector.assign\_operator\_change\_vector\_size  [ OK ] TDynamicVector.assign\_operator\_change\_vector\_size (0 ms)  [ RUN ] TDynamicVector.compare\_equal\_vectors\_return\_true  [ OK ] TDynamicVector.compare\_equal\_vectors\_return\_true (0 ms)  [ RUN ] TDynamicVector.compare\_vector\_with\_itself\_return\_true  [ OK ] TDynamicVector.compare\_vector\_with\_itself\_return\_true (0 ms)  [ RUN ] TDynamicVector.vectors\_with\_different\_size\_are\_not\_equal  [ OK ] TDynamicVector.vectors\_with\_different\_size\_are\_not\_equal (1 ms)  [ RUN ] TDynamicVector.cant\_multiply\_vectors\_with\_not\_equal\_size  [ OK ] TDynamicVector.cant\_multiply\_vectors\_with\_not\_equal\_size (0 ms)  [ RUN ] TDynamicVector.can\_resize\_vector  [ OK ] TDynamicVector.can\_resize\_vector (0 ms)  [ RUN ] TDynamicVector.cant\_resize\_vector\_to\_negative\_size  [ OK ] TDynamicVector.cant\_resize\_vector\_to\_negative\_size (0 ms)  [ RUN ] TDynamicVector.cant\_found\_element\_with\_negative\_index  [ OK ] TDynamicVector.cant\_found\_element\_with\_negative\_index (0 ms)  [ RUN ] TDynamicVector.can\_found\_element\_with\_positive\_index  [ OK ] TDynamicVector.can\_found\_element\_with\_positive\_index (0 ms)  [ RUN ] TDynamicVector.can\_add\_scalar\_to\_vector  [ OK ] TDynamicVector.can\_add\_scalar\_to\_vector (0 ms)  [ RUN ] TDynamicVector.can\_substract\_scalar\_from\_vector  [ OK ] TDynamicVector.can\_substract\_scalar\_from\_vector (0 ms)  [ RUN ] TDynamicVector.can\_multiply\_vector\_by\_scalar  [ OK ] TDynamicVector.can\_multiply\_vector\_by\_scalar (0 ms)  [ RUN ] TDynamicVector.can\_sum\_vectors  [ OK ] TDynamicVector.can\_sum\_vectors (0 ms)  [ RUN ] TDynamicVector.can\_substact\_vectors  [ OK ] TDynamicVector.can\_substact\_vectors (0 ms)  [ RUN ] TDynamicVector.can\_equate\_vector  [ OK ] TDynamicVector.can\_equate\_vector (0 ms)  [----------] 20 tests from TDynamicVector (21 ms total)  [----------] Global test environment tear-down  [==========] 54 tests from 3 test cases ran. (48 ms total)  [ PASSED ] 54 tests. |

Заключение

В ходе лабораторной работы мною была выполнена поставленная задача. Мне удалось написать классы для работы с векторами и матрицами, используя шаблоны. Также работоспособность была проверена с помощью тестов и примеров.

Литература

1. Bjarne Stroustrup: The C++ Programming Language. Addison-Wesley. 1986.
2. Топп У., Форд У. Структуры данных в С++. –М. Бином, 1999.
3. Шилдт Г. Самоучитель С++: Пер. с англ. -3-е изд. –СПб.: БХВ-Петербург, 2006.

Приложения

Приложение 1

|  |
| --- |
| #ifndef \_\_TDynamicVector\_H\_\_  #define \_\_TDynamicVector\_H\_\_  #include <iostream>  using namespace std;  template<class T>  class TDynamicVector  {  protected:  int len;  T\* pMem;  public:  TDynamicVector();  TDynamicVector(int n);  TDynamicVector(T\* arr, int s);  TDynamicVector(const TDynamicVector<T>& v);  TDynamicVector(TDynamicVector<T>&& v) noexcept;  ~TDynamicVector();  int size();  void Resize(int nlen);  T& operator[](int i);  const T& operator[](int i) const;  bool operator==(const TDynamicVector<T>& v);  bool operator!=(const TDynamicVector<T>& v);  TDynamicVector<T> operator+(T n);  TDynamicVector<T> operator-(double n);  TDynamicVector<T> operator\*(double n);  TDynamicVector<T> operator+(const TDynamicVector<T>& v);  TDynamicVector<T> operator-(const TDynamicVector<T>& v);  TDynamicVector<T> operator\*(const TDynamicVector<T>& v);  TDynamicVector<T>& operator=(const TDynamicVector<T>& v);  TDynamicVector<T>& operator=(TDynamicVector<T>&& v) noexcept;  friend istream& operator>>(istream& istr, const TDynamicVector<T>& v);  friend ostream& operator<<(ostream& ostr, const TDynamicVector<T>& v);  };  #endif  template<class T>  TDynamicVector<T>::TDynamicVector()  {  len = NULL;  pMem = nullptr;  }  template<class T>  inline TDynamicVector<T>::TDynamicVector(int n)  {  if (n <= 0) throw "error";  len = n;  pMem = new T[len];  for (int i = 0; i < len; i++)  pMem[i] = 1;  }  template<class T>  inline TDynamicVector<T>::TDynamicVector(T\* arr, int s)  {  len = s;  pMem = new T[len];  for (int i = 0; i < len; i++)  pMem[i] = arr[i];  }  template<class T>  inline TDynamicVector<T>::TDynamicVector(const TDynamicVector<T>& v)  {  len = v.len;  if (v.len == 0)  pMem = nullptr;  else  pMem = new T[len];  for (int i = 0; i < len; i++)  pMem[i] = v.pMem[i];  }  template<class T>  inline TDynamicVector<T>::TDynamicVector(TDynamicVector<T>&& v) noexcept  {  pMem = v.pMem;  len = v.len;  v.pMem = nullptr;  v.len = 0;  }  template<class T>  inline TDynamicVector<T>::~TDynamicVector()  {  if (pMem != nullptr)  {  delete[] pMem;  pMem = nullptr;  }  len = NULL;  }  template<class T>  inline int TDynamicVector<T>::size()  {  return len;  }  template<class T>  inline void TDynamicVector<T>::Resize(int nlen)  {  if (nlen < 0) throw "error";  TDynamicVector<T> res(\*this);  delete[] pMem;  pMem = new T[nlen];  for (int i = 0; i < min(nlen, res.len); i++)  {  pMem[i] = res.pMem[i];  }  if (nlen > res.len)  for (int j = res.len; j < nlen; j++)  pMem[j] = 0;  len = nlen;  }  template<class T>  inline T& TDynamicVector<T>::operator[](int i)  {  if (len == 0) throw "error";  if ((i < 0) || (i > len)) throw "not found";  return pMem[i];  }  template<class T>  inline const T& TDynamicVector<T>::operator[](int i) const  {  if (len == 0) throw "error";  if ((i < 0) || (i > len)) throw "not found";  return pMem[i];  }  template<class T>  inline bool TDynamicVector<T>::operator==(const TDynamicVector<T>& v)  {  if (len != v.len) return false;  else  {  for (int i = 0; i < len; i++)  if (pMem[i] != v.pMem[i])  return false;  }  return true;  }  template<class T>  inline bool TDynamicVector<T>::operator!=(const TDynamicVector<T>& v)  {  if (len != v.len) return true;  else  {  for (int i = 0; i < len; i++)  if (pMem[i] != v.pMem[i])  return true;  }  return false;  }  template<class T>  inline TDynamicVector<T> TDynamicVector<T>::operator+(T n)  {  if (n == 0) return \*this;  TDynamicVector<T> res(\*this);  {  for (int i = 0; i < len; i++)  res.pMem[i] = res.pMem[i] + n;  }  return res;  }  template<class T>  inline TDynamicVector<T> TDynamicVector<T>::operator-(double n)  {  if (n == 0) return \*this;  TDynamicVector<T> res(\*this);  {  for (int i = 0; i < len; i++)  res.pMem[i] = res.pMem[i] - n;  }  return res;  }  template<class T>  inline TDynamicVector<T> TDynamicVector<T>::operator\*(double n)  {  if (n == 0) pMem = nullptr;  TDynamicVector<T> res(\*this);  {  for (int i = 0; i < len; i++)  res.pMem[i] = res.pMem[i] \* n;  }  return res;  }  template<class T>  inline TDynamicVector<T> TDynamicVector<T>::operator+(const TDynamicVector<T>& v)  {  TDynamicVector<T> res(\*this);  if ((len == v.len) && (v.pMem != 0) && (pMem != 0))  {  for (int i = 0; i < len; i++)  res.pMem[i] = res.pMem[i] + v.pMem[i];  }  else  throw "error";  return res;  }  template<class T>  inline TDynamicVector<T> TDynamicVector<T>::operator-(const TDynamicVector<T>& v)  {  TDynamicVector<T> res(\*this);  if ((len == v.len) && (v.pMem != 0) && (pMem != 0))  {  for (int i = 0; i < len; i++)  res.pMem[i] = res.pMem[i] - v.pMem[i];  }  else  throw "error";  return res;  }  template<class T>  inline TDynamicVector<T> TDynamicVector<T>::operator\*(const TDynamicVector<T>& v)  {  TDynamicVector<T> res(\*this);  if ((len == v.len) && (v.pMem != 0) && (pMem != 0))  {  for (int i = 0; i < len; i++)  res.pMem[i] = res.pMem[i] \* v.pMem[i];  }  else  throw "error";  return res;  }  template<class T>  inline TDynamicVector<T>& TDynamicVector<T>::operator=(const TDynamicVector<T>& v)  {  if (this == &v) return \*this;  if (v.pMem == 0)  {  delete[] pMem;  pMem = 0;  len = 0;  return \*this;  }  if (pMem != 0)  delete[] pMem;  len = v.len;  pMem = new T[len];  for (int i = 0; i < len; i++)  pMem[i] = v.pMem[i];  return \*this;  }  template<class T>  inline TDynamicVector<T>& TDynamicVector<T>::operator=(TDynamicVector<T>&& v) noexcept  {  if (this != &v)  {  delete[] pMem;  pMem = v.pMem;  len = v.len;  v.pMem = nullptr;  v.len = 0;  }  return \*this;  }  template <class T>  std::ostream& operator<<(std::ostream& ostr, TDynamicVector<T>& v)  {  for (int i = 0; i < v.size(); i++)  ostr << v[i] << ' ';  return ostr;  }  template <class T>  std::istream& operator>>(std::istream& istr, TDynamicVector<T>& v)  {  T a = 0;  std::cout << "Enter your vector = " << std::endl;  for (int i = 0; i < v.size(); i++)  {  istr >> a;  v[i] = a;  }  return istr;  } |

Приложение 2

|  |
| --- |
| #ifndef \_\_TSquareDynamicMatrix\_H\_\_  #define \_\_TSquareDynamicMatrix\_H\_\_  #include <iostream>  #include "TTriangleMatrix.h"  using namespace std;  template<class T>  class TSquareDynamicMatrix : protected TDynamicVector<TDynamicVector<T>>  {  using TDynamicVector<TDynamicVector<T>>::pMem;  using TDynamicVector<TDynamicVector<T>>::len;  public:  using TDynamicVector<TDynamicVector<T>>::operator[];  using TDynamicVector<TDynamicVector<T>>::size;  TSquareDynamicMatrix();  TSquareDynamicMatrix(int n);  TSquareDynamicMatrix(const TSquareDynamicMatrix<T>& m);  bool operator==(const TSquareDynamicMatrix<T>& m);  bool operator!=(const TSquareDynamicMatrix<T>& m);  TSquareDynamicMatrix<T> operator\*(const T& v);  TDynamicVector<T> operator\*(TDynamicVector<T>& v);  TSquareDynamicMatrix<T> operator+(const TSquareDynamicMatrix<T>& m);  TSquareDynamicMatrix<T> operator-(const TSquareDynamicMatrix<T>& m);  TSquareDynamicMatrix<T> operator\*(const TSquareDynamicMatrix<T>& m);  TSquareDynamicMatrix<T> operator=(const TSquareDynamicMatrix<T>& m);  friend istream& operator>>(istream& istr, const TSquareDynamicMatrix<T>& m);  friend ostream& operator<<(ostream& ostr, const TSquareDynamicMatrix<T>& m);  };  #endif  template<class T>  inline TSquareDynamicMatrix<T>::TSquareDynamicMatrix()  {  this->len = NULL;  this->pMem = nullptr;  }  template<class T>  inline TSquareDynamicMatrix<T>::TSquareDynamicMatrix(int n)  {  if (n <= 0) throw "error";  this->len = n;  if (this->pMem != nullptr)  delete[] this->pMem;  this->pMem = new TDynamicVector<T>[this->len];  for (int i = 0; i < this->len; i++)  this->pMem[i] = TDynamicVector<T>(len);  }  template<class T>  inline TSquareDynamicMatrix<T>::TSquareDynamicMatrix(const TSquareDynamicMatrix<T>& m)  {  if (m.len == 0)  this->pMem = 0;  if (m.len > 0)  this->len = m.len;  this->pMem = new TDynamicVector<T>[len];  for (int i = 0; i < len; i++)  this->pMem[i] = m.pMem[i];  }  template<class T>  inline bool TSquareDynamicMatrix<T>::operator==(const TSquareDynamicMatrix<T>& m)  {  return TDynamicVector < TDynamicVector<T>> :: operator==(m);  }  template<class T>  inline bool TSquareDynamicMatrix<T>::operator!=(const TSquareDynamicMatrix<T>& m)  {  return TDynamicVector < TDynamicVector<T>> :: operator!=(m);  }  template<class T>  inline TSquareDynamicMatrix<T> TSquareDynamicMatrix<T>::operator\*(const T& v)  {  TSquareDynamicMatrix<T> res(\*this);  for (int i = 0; i < len; i++)  res.pMem[i] = pMem[i] \* v;  return res;  }  template<class T>  inline TDynamicVector<T> TSquareDynamicMatrix<T>::operator\*(TDynamicVector<T>& v)  {  if (this->len != v.size()) throw "error";  TDynamicVector<T> res(this->len);  for (int i = 0; i < len; i++)  {  res[i] = 0;  for (int j = 0; j < len; j++)  res[i] += this->pMem[i][j] \* v[j];  }  return res;  }  template<class T>  inline TSquareDynamicMatrix<T> TSquareDynamicMatrix<T>::operator+(const TSquareDynamicMatrix<T>& m)  {  if (m.len != len) throw "error";  TSquareDynamicMatrix<T> res(\*this);  for (int i = 0; i < len; i++)  res.pMem[i] = pMem[i] + m.pMem[i];  return res;  }  template<class T>  inline TSquareDynamicMatrix<T> TSquareDynamicMatrix<T>::operator-(const TSquareDynamicMatrix<T>& m)  {  if (m.len != len) throw "error";  TSquareDynamicMatrix<T> res(\*this);  for (int i = 0; i < len; i++)  res.pMem[i] = pMem[i] - m.pMem[i];  return res;  }  template<class T>  inline TSquareDynamicMatrix<T> TSquareDynamicMatrix<T>::operator\*(const TSquareDynamicMatrix<T>& m)  {  if (m.len != len) throw "error";  TSquareDynamicMatrix<T> res(\*this);  for (int i = 0; i < len; i++)  for (int j = 0; j < len; j++)  {  res[i][j] = 0;  for (int k = 0; k < len; k++)  res.pMem[i][j] += pMem[i][k] \* m.pMem[k][j];  }  return res;  }  template<class T>  inline TSquareDynamicMatrix<T> TSquareDynamicMatrix<T>::operator=(const TSquareDynamicMatrix<T>& m)  {  TDynamicVector<TDynamicVector<T>>::operator=(m);  return \*this;  }  template<class T>  std::istream& operator>>(std::istream& istr, TSquareDynamicMatrix<T>& m)  {  std::cout << "Enter your matrix = " << std::endl;  for (int i = 0; i < m.size(); i++)  {  for (int j = 0; j < i + 1; j++)  {  istr >> m[i][j];  }  }  return istr;  }  template<class T>  std::ostream& operator<<(std::ostream& ostr, TSquareDynamicMatrix<T>& m)  {  for (int i = 0; i < m.size(); i++)  {  for (int j = 0; j < m.size(); j++)  {  ostr << m[i][j] << ' ';  }  std::cout << std::endl;  }  return ostr;  } |

Приложение 3

|  |
| --- |
| #ifndef \_\_TTriangleDynamicMatrix\_H\_\_  #define \_\_TTriangleDynamicMatrix\_H\_\_  #include <iostream>  #include "TVector.h"  using namespace std;  template<class T>  class TTriangleDynamicMatrix : protected TDynamicVector<TDynamicVector<T>>  {  using TDynamicVector<TDynamicVector<T>>::pMem;  using TDynamicVector<TDynamicVector<T>>::len;  public:  using TDynamicVector<TDynamicVector<T>>::operator[];  using TDynamicVector<TDynamicVector<T>>::size;  TTriangleDynamicMatrix();  TTriangleDynamicMatrix(int n);  TTriangleDynamicMatrix(const TTriangleDynamicMatrix<T>& m);  bool operator==(const TTriangleDynamicMatrix<T>& m);  bool operator!=(const TTriangleDynamicMatrix<T>& m);  TTriangleDynamicMatrix<T> operator\*(const T& v);  TDynamicVector<T> operator\*(TDynamicVector<T>& v);  TTriangleDynamicMatrix<T> operator+(const TTriangleDynamicMatrix<T>& m);  TTriangleDynamicMatrix<T> operator-(const TTriangleDynamicMatrix<T>& m);  TTriangleDynamicMatrix<T> operator\*(const TTriangleDynamicMatrix<T>& m);  TTriangleDynamicMatrix<T> operator=(const TTriangleDynamicMatrix<T>& m);  friend istream& operator>>(istream& istr, const TTriangleDynamicMatrix<T>& m);  friend ostream& operator<<(ostream& ostr, const TTriangleDynamicMatrix<T>& m);  };  #endif  template<class T>  inline TTriangleDynamicMatrix<T>::TTriangleDynamicMatrix()  {  this->len = NULL;  this->pMem = nullptr;  }  template<class T>  inline TTriangleDynamicMatrix<T>::TTriangleDynamicMatrix(int n)  {  if (n <= 0) throw "error";  this->len = n;  if (this->pMem != nullptr)  delete[] this->pMem;  this->pMem = new TDynamicVector<T>[this->len];  for (int i = 0; i < this->len; i++)  this->pMem[i] = TDynamicVector<T>(i + 1);  }  template<class T>  inline TTriangleDynamicMatrix<T>::TTriangleDynamicMatrix(const TTriangleDynamicMatrix<T>& m)  {  if (m.len == 0)  this->pMem = 0;  if (m.len > 0)  this->len = m.len;  this->pMem = new TDynamicVector<T>[len];  for (int i = 0; i < len; i++)  this->pMem[i] = m.pMem[i];  }  template<class T>  inline bool TTriangleDynamicMatrix<T>::operator==(const TTriangleDynamicMatrix<T>& m)  {  return TDynamicVector < TDynamicVector<T>> :: operator==(m);  }  template<class T>  inline bool TTriangleDynamicMatrix<T>::operator!=(const TTriangleDynamicMatrix<T>& m)  {  return TDynamicVector < TDynamicVector<T>> :: operator!=(m);  }  template<class T>  inline TTriangleDynamicMatrix<T> TTriangleDynamicMatrix<T>::operator\*(const T& v)  {  TTriangleDynamicMatrix<T> res(\*this);  for (int i = 0; i < len; i++)  res.pMem[i] = pMem[i] \* v;  return res;  }  template<class T>  inline TDynamicVector<T> TTriangleDynamicMatrix<T>::operator\*(TDynamicVector<T>& v)  {  if (this->len != v.size()) throw "error";  TDynamicVector<T> res(this->len);  for (int i = 0; i < len; i++)  {  res[i] = 0;  for (int j = 0; j <= i; j++)  res[j] += this->pMem[i][j] \* v[i];  }  return res;  }  template<class T>  inline TTriangleDynamicMatrix<T> TTriangleDynamicMatrix<T>::operator+(const TTriangleDynamicMatrix<T>& m)  {  if (m.len != len) throw "error";  TTriangleDynamicMatrix<T> res(\*this);  for (int i = 0; i < len; i++)  res.pMem[i] = pMem[i] + m.pMem[i];  return res;  }  template<class T>  inline TTriangleDynamicMatrix<T> TTriangleDynamicMatrix<T>::operator-(const TTriangleDynamicMatrix<T>& m)  {  if (m.len != len) throw "error";  TTriangleDynamicMatrix<T> res(\*this);  for (int i = 0; i < len; i++)  res.pMem[i] = pMem[i] - m.pMem[i];  return res;  }  template<class T>  inline TTriangleDynamicMatrix<T> TTriangleDynamicMatrix<T>::operator\*(const TTriangleDynamicMatrix<T>& m)  {  if (m.len != len) throw "error";  TTriangleDynamicMatrix<T> res(\*this);  for (int i = 0; i < len; i++)  for (int j = 0; j <= i; j++)  {  res[i][j] = 0;  for (int k = j; k <= i; k++)  res.pMem[i][j] += m.pMem[i][k] \* pMem[k][j];  }  return res;  }  template<class T>  inline TTriangleDynamicMatrix<T> TTriangleDynamicMatrix<T>::operator=(const TTriangleDynamicMatrix<T>& m)  {  TDynamicVector<TDynamicVector<T>>::operator=(m);  return \*this;  }  template<class T>  std::istream& operator>>(std::istream& istr, TTriangleDynamicMatrix<T>& m)  {  std::cout << "Enter your matrix = " << std::endl;  for (int i = 0; i < m.size(); i++)  {  for (int j = 0; j < i + 1; j++)  {  istr >> m[i][j];  }  }  return istr;  }  template<class T>  std::ostream& operator<<(std::ostream& ostr, TTriangleDynamicMatrix<T>& m)  {  for (int i = 0; i < m.size(); i++)  {  for (int j = 0; j < i + 1; j++)  {  ostr << m[i][j] << ' ';  }  std::cout << std::endl;  }  return ostr;  } |

Приложение 4

|  |
| --- |
| #include "gtest.h"  #include "TTriangleMatrix.h"  TEST(TDynamicVector, can\_create\_vector\_with\_positive\_length)  {  ASSERT\_NO\_THROW(TDynamicVector<int> v(5));  }  TEST(TDynamicVector, throws\_when\_create\_vector\_with\_negative\_length)  {  ASSERT\_ANY\_THROW(TDynamicVector<int> v(-5));  }  TEST(TDynamicVector, can\_create\_copied\_vector)  {  TDynamicVector<int> v(10);  ASSERT\_NO\_THROW(TDynamicVector<int> v1(v));  }  TEST(TDynamicVector, copied\_vector\_has\_its\_own\_memory)  {  TDynamicVector<int> v1(4);  v1[2] = 2;  TDynamicVector<int> v2 = v1;  EXPECT\_NE(&v1, &v2);  }  TEST(TDynamicVector, can\_get\_size)  {  TDynamicVector<int> v(4);  EXPECT\_EQ(4, v.size());  }  TEST(TDynamicVector, assign\_operator\_change\_vector\_size)  {  TDynamicVector<int> v1(2), v2(4);  v2 = v1;  EXPECT\_EQ(v1.size(), v2.size());  }  TEST(TDynamicVector, compare\_equal\_vectors\_return\_true)  {  TDynamicVector<int> v1(2), v2(2);  v1 = v2;  EXPECT\_TRUE(v1 == v2);  }  TEST(TDynamicVector, compare\_vector\_with\_itself\_return\_true)  {  TDynamicVector<int> v1(2);  v1[0] = 2;  v1[1] = 2;  EXPECT\_TRUE(v1 == v1);  }  TEST(TDynamicVector, vectors\_with\_different\_size\_are\_not\_equal)  {  TDynamicVector<int> v1(2), v2(4);  EXPECT\_TRUE(v1 != v2);  }  TEST(TDynamicVector, cant\_multiply\_vectors\_with\_not\_equal\_size)  {  TDynamicVector<int> v1(2), v2(4);  ASSERT\_ANY\_THROW(v1 \* v2);  }  TEST(TDynamicVector, can\_resize\_vector)  {  TDynamicVector<int> v(5);  v.Resize(8);  EXPECT\_EQ(8, v.size());  }  TEST(TDynamicVector, cant\_resize\_vector\_to\_negative\_size)  {  TDynamicVector<int> v1(2);  ASSERT\_ANY\_THROW(v1.Resize(-2));  }  TEST(TDynamicVector, cant\_found\_element\_with\_negative\_index)  {  TDynamicVector<int> v1(2);  ASSERT\_ANY\_THROW(v1[-1]);  }  TEST(TDynamicVector, can\_found\_element\_with\_positive\_index)  {  TDynamicVector<int> v1(2);  ASSERT\_NO\_THROW(v1[1]);  }  TEST(TDynamicVector, can\_add\_scalar\_to\_vector)  {  TDynamicVector<int> v1(2);  int b = 4;  ASSERT\_NO\_THROW(v1 + b);  }  TEST(TDynamicVector, can\_substract\_scalar\_from\_vector)  {  TDynamicVector<int> v1(2);  double b = 4.0;  ASSERT\_NO\_THROW(v1 - b);  }  TEST(TDynamicVector, can\_multiply\_vector\_by\_scalar)  {  TDynamicVector<int> v1(2);  v1[0] = 2;  v1[1] = 2;  double b = 4.0;  TDynamicVector<int> v2(2);  v2[0] = 8;  v2[1] = 8;  ASSERT\_NO\_THROW(v1 \* b);  EXPECT\_TRUE(v2 == (v1 \* b));  }  TEST(TDynamicVector, can\_sum\_vectors)  {  TDynamicVector<int> v1(2);  v1[0] = 2;  v1[1] = 2;  TDynamicVector<int> v2(2);  v2[0] = 8;  v2[1] = 8;  ASSERT\_NO\_THROW(v1 + v2);  }  TEST(TDynamicVector, can\_substact\_vectors)  {  TDynamicVector<int> v1(2);  v1[0] = 2;  v1[1] = 2;  TDynamicVector<int> v2(2);  v2[0] = 8;  v2[1] = 8;  ASSERT\_NO\_THROW(v1 - v2);  }  TEST(TDynamicVector, can\_equate\_vector)  {  TDynamicVector<int> v1(2);  v1[0] = 2;  v1[1] = 2;  TDynamicVector<int> v2(2);  v2[0] = 8;  v2[1] = 8;  v2 = v1;  EXPECT\_TRUE(v2 == v1);  EXPECT\_EQ(v2[0], 2);  } |

Приложение 5

|  |
| --- |
| #include "gtest.h"  #include "TSquareMatrix.h"  TEST(TTriangleDynamicMatrix, can\_create\_matrix\_with\_positive\_length)  {  ASSERT\_NO\_THROW(TTriangleDynamicMatrix<int> m(5));  }  TEST(TTriangleDynamicMatrix, throws\_when\_create\_matrix\_with\_negative\_length)  {  ASSERT\_ANY\_THROW(TTriangleDynamicMatrix<int> m(-5));  }  TEST(TTriangleDynamicMatrix, can\_create\_copied\_matrix)  {  TTriangleDynamicMatrix<int> m(5);  ASSERT\_NO\_THROW(TTriangleDynamicMatrix<int> m1(m));  }  TEST(TTriangleDynamicMatrix, can\_get\_size)  {  TTriangleDynamicMatrix<int> m(4);  EXPECT\_EQ(4, m.size());  }  TEST(TTriangleDynamicMatrix, compare\_equal\_matrices\_return\_true)  {  TTriangleDynamicMatrix<int> m1(2), m2(2);  m1 = m2;  EXPECT\_TRUE(m1 == m2);  }  TEST(TTriangleDynamicMatrix, compare\_matrix\_with\_itself\_return\_true)  {  TTriangleDynamicMatrix<int> m1(2);  EXPECT\_TRUE(m1 == m1);  }  TEST(TTriangleDynamicMatrix, matrices\_with\_different\_size\_are\_not\_equal)  {  TTriangleDynamicMatrix<int> m1(2), m2(4);  EXPECT\_TRUE(m1 != m2);  }  TEST(TTriangleDynamicMatrix, cant\_add\_matrices\_with\_not\_equal\_size)  {  TTriangleDynamicMatrix<int> m1(2), m2(4);  ASSERT\_ANY\_THROW(m1 + m2);  }  TEST(TTriangleDynamicMatrix, can\_add\_matrices\_with\_equal\_size)  {  TTriangleDynamicMatrix<int> m1(2), m2(2), m3(2);  m1[0][0] = 1;  m1[1][0] = 1;  m1[1][1] = 1;  m2[0][0] = 2;  m2[1][0] = 2;  m2[1][1] = 2;  m3 = m1 + m2;  EXPECT\_EQ(m3[0][0], 3);  }  TEST(TTriangleDynamicMatrix, cant\_subtract\_matrixes\_with\_not\_equal\_size)  {  TTriangleDynamicMatrix<int> m1(2), m2(4);  ASSERT\_ANY\_THROW(m1 - m2);  }  TEST(TTriangleDynamicMatrix, can\_subtract\_matrixes\_with\_equal\_size)  {  TTriangleDynamicMatrix<int> m1(2), m2(2), m3(2);  m1[0][0] = 1;  m1[1][0] = 1;  m1[1][1] = 1;  m2[0][0] = 2;  m2[1][0] = 2;  m2[1][1] = 2;  m3 = m1 - m2;  ASSERT\_NO\_THROW(m1 - m2);  EXPECT\_EQ(m3[0][0], -1);  }  TEST(TTriangleDynamicMatrix, cant\_multiply\_matrixes\_with\_not\_equal\_size)  {  TTriangleDynamicMatrix<int> m1(2), m2(4);  ASSERT\_ANY\_THROW(m1 \* m2);  }  TEST(TTriangleDynamicMatrix, can\_equate\_matrixes)  {  TTriangleDynamicMatrix<int> m1(2), m2(4);  ASSERT\_NO\_THROW(m1 = m2);  }  TEST(TTriangleDynamicMatrix, can\_multiply\_matrixes\_with\_equal\_size)  {  TTriangleDynamicMatrix<int> m1(2), m2(2), m3(2);  m1[0][0] = 1;  m1[1][0] = 1;  m1[1][1] = 1;  m2[0][0] = 2;  m2[1][0] = 2;  m2[1][1] = 2;  m3 = m1 \* m2;  EXPECT\_EQ(m3[0][0], 2);  EXPECT\_EQ(m3[1][0], 4);  }  TEST(TTriangleDynamicMatrix, can\_multiply\_matrix\_by\_vector)  {  TTriangleDynamicMatrix<int> m1(2);  TDynamicVector<int> v(2), vres;  m1[0][0] = 1;  m1[1][0] = 2;  m1[1][1] = 4;  v[0] = 1;  v[1] = 2;  vres = m1 \* v;  EXPECT\_EQ(vres[0], 5);  EXPECT\_EQ(vres[1], 8);  }  TEST(TTriangleDynamicMatrix, cant\_multiply\_matrix\_by\_vector\_with\_not\_equal)  {  TTriangleDynamicMatrix<int> m1(2);  TDynamicVector<int> v(3);  ASSERT\_ANY\_THROW(m1 \* v);  }  TEST(TTriangleDynamicMatrix, can\_multiply\_matrix\_by\_scalar)  {  TTriangleDynamicMatrix<int> mat(2);  mat[0][0] = 2;  mat[1][0] = 3;  mat[1][1] = 5;  TTriangleDynamicMatrix<int> res = mat \* 2;  EXPECT\_EQ(res[0][0], 4);  EXPECT\_EQ(res[1][0], 6);  EXPECT\_EQ(res[1][1], 10);  }  TEST(TSquareDynamicMatrix, can\_create\_matrix\_with\_positive\_length)  {  ASSERT\_NO\_THROW(TSquareDynamicMatrix<int> m(5));  }  TEST(TSquareDynamicMatrix, throws\_when\_create\_matrix\_with\_negative\_length)  {  ASSERT\_ANY\_THROW(TSquareDynamicMatrix<int> m(-5));  }  TEST(TSquareDynamicMatrix, can\_create\_copied\_matrix)  {  TSquareDynamicMatrix<int> m(5);  ASSERT\_NO\_THROW(TSquareDynamicMatrix<int> m1(m));  }  TEST(TSquareDynamicMatrix, can\_get\_size)  {  TSquareDynamicMatrix<int> m(4);  EXPECT\_EQ(4, m.size());  }  TEST(TSquareDynamicMatrix, compare\_equal\_matrices\_return\_true)  {  TSquareDynamicMatrix<int> m1(2), m2(2);  m1 = m2;  EXPECT\_TRUE(m1 == m2);  }  TEST(TSquareDynamicMatrix, compare\_matrix\_with\_itself\_return\_true)  {  TSquareDynamicMatrix<int> m1(2);  EXPECT\_TRUE(m1 == m1);  }  TEST(TSquareDynamicMatrix, matrices\_with\_different\_size\_are\_not\_equal)  {  TSquareDynamicMatrix<int> m1(2), m2(4);  EXPECT\_TRUE(m1 != m2);  }  TEST(TSquareDynamicMatrix, cant\_add\_matrices\_with\_not\_equal\_size)  {  TSquareDynamicMatrix<int> m1(2), m2(4);  ASSERT\_ANY\_THROW(m1 + m2);  }  TEST(TSquareDynamicMatrix, can\_add\_matrices\_with\_equal\_size)  {  TSquareDynamicMatrix<int> m1(2), m2(2), m3(2);  m1[0][0] = 1;  m1[1][0] = 1;  m1[1][1] = 1;  m2[0][0] = 2;  m2[1][0] = 2;  m2[1][1] = 2;  m3 = m1 + m2;  EXPECT\_EQ(m3[0][0], 3);  }  TEST(TSquareDynamicMatrix, cant\_subtract\_matrixes\_with\_not\_equal\_size)  {  TSquareDynamicMatrix<int> m1(2), m2(4);  ASSERT\_ANY\_THROW(m1 - m2);  }  TEST(TSquareDynamicMatrix, can\_subtract\_matrixes\_with\_equal\_size)  {  TSquareDynamicMatrix<int> m1(2), m2(2), m3(2);  m1[0][0] = 1;  m1[1][0] = 1;  m1[1][1] = 1;  m2[0][0] = 2;  m2[1][0] = 2;  m2[1][1] = 2;  m3 = m1 - m2;  ASSERT\_NO\_THROW(m1 - m2);  EXPECT\_EQ(m3[0][0], -1);  }  TEST(TSquareDynamicMatrix, cant\_multiply\_matrixes\_with\_not\_equal\_size)  {  TSquareDynamicMatrix<int> m1(2), m2(4);  ASSERT\_ANY\_THROW(m1 \* m2);  }  TEST(TSquareDynamicMatrix, can\_equate\_matrixes)  {  TSquareDynamicMatrix<int> m1(2), m2(4);  ASSERT\_NO\_THROW(m1 = m2);  }  TEST(TSquareDynamicMatrix, can\_multiply\_matrixes\_with\_equal\_size)  {  TSquareDynamicMatrix<int> m1(2), m2(2), m3(2);  m1[0][0] = 1;  m1[0][1] = 1;  m1[1][0] = 1;  m1[1][1] = 1;  m2[0][0] = 2;  m2[0][1] = 2;  m2[1][0] = 2;  m2[1][1] = 2;  m3 = m1 \* m2;  EXPECT\_EQ(m3[0][0], 4);  EXPECT\_EQ(m3[1][0], 4);  }  TEST(TSquareDynamicMatrix, can\_multiply\_matrix\_by\_vector)  {  TSquareDynamicMatrix<int> m1(2);  TDynamicVector<int> v(2), vres;  m1[0][0] = 1;  m1[0][1] = 3;  m1[1][0] = 2;  m1[1][1] = 4;  v[0] = 1;  v[1] = 2;  vres = m1 \* v;  EXPECT\_EQ(vres[0], 7);  EXPECT\_EQ(vres[1], 10);  }  TEST(TSquareDynamicMatrix, cant\_multiply\_matrix\_by\_vector\_with\_not\_equal)  {  TSquareDynamicMatrix<int> m1(2);  TDynamicVector<int> v(3);  ASSERT\_ANY\_THROW(m1 \* v);  }  TEST(TSquareDynamicMatrix, can\_multiply\_matrix\_by\_scalar)  {  TSquareDynamicMatrix<int> mat(2);  mat[0][0] = 2;  mat[1][0] = 3;  mat[1][1] = 5;  TSquareDynamicMatrix<int> res = mat \* 2;  EXPECT\_EQ(res[0][0], 4);  EXPECT\_EQ(res[1][0], 6);  EXPECT\_EQ(res[1][1], 10);  } |