Федеральное государственное автономное образовательное учреждение высшего образования **"Национальный исследовательский Нижегородский государственный университет им. Н.И. Лобачевского" (ННГУ).**

**Институт информационных технологий, математики и механики**

**ОТЧЕТ**

По лабораторной работе

На тему:

«Динамические вектора и матрицы»

**Выполнила:**

Студентка группы 3821Б1ПМ3

Киселева К.В.

**Проверил:**

Заведующий лабораторией суперкомпьютерных технологий и высокопроизводительных вычислений

Лебедев И.Г.

Нижний Новгород

2022 г

Оглавление

[Введение. 3](#_Toc119159795)

[1. Постановка задачи. 4](#_Toc119159796)

[2. Руководство пользователя. 5](#_Toc119159797)

[2.1 Библиотека «Vector.h» 5](#_Toc119159798)

[2.2 Библиотека «Matrix.h» 6](#_Toc119159799)

[3. Руководство программиста. 8](#_Toc119159800)

[3.1 Описание структуры программы. 8](#_Toc119159801)

[3.2 Описание алгоритмов. 8](#_Toc119159802)

[3.3 Теоретическая асимптотическая точность 10](#_Toc119159803)

[4. Эксперименты. 12](#_Toc119159804)

[Литература. 14](#_Toc119159805)

[Приложение. 15](#_Toc119159806)

[Приложение 1. 15](#_Toc119159807)

# Введение.

Программирование - это интересный, полезный и увлекательный процесс, благодаря которому мы, с помощью специальных команд, заставляем компьютер, выполнять для нас различные задачи, от выполнения операций с числами и навигации, до управления самолетами, спутниками и прочей техникой.

Случайные числа являются одной из основных составляющих любого языка программирования, на них строятся различные алгоритмы.

В данной лабораторной работе для изучения методов и особенностей работы со случайными числами была поставлена задача, которую нужно было выполнить, используя язык программирования «С».

# Постановка задачи.

Написать классы для работы с векторами и матрицами, использовать шаблоны. Вектора в математическом понимании: имеется набор значений из N мерного пространства, размерность задается как параметр. Классы вектора и матрицы должны быть вынесены в статическую библиотеку.

# Руководство пользователя.

Были созданы две статические библиотеки, одна – для работы с векторами, вторая ­– для работы с матрицами.

## Библиотека «Vector.h»

В этой библиотеке представлены стандартные конструкторы, деструктор, перегруженные операторы и методы.

Конструкторы и деструктор. (см. Таблица 1)

|  |  |
| --- | --- |
| TVector(); | Используя этот конструктор, получим вектор, имеющий нулевую длинну и не содержащий никаких данных. |
| TVector(const int n); | Используя этот конструктор, получим вектор длины n, каждая координата которого изначально равна 0 |
| TVector(const int n, const T\* \_data); | Используя этот конструктор, получим вектор длины n, координаты будут равны соответствующим значениям из массива, который мы передаём. |
| TVector(const int \_len, const T a); | Используя этот конструктор, получим вектор длины \_len, каждая координата которого изначально равна значению числа a. |
| TVector(const TVector<T>& a); | Используя этот конструктор, получим вектор, равный вектору a. |
| ~TVector(); | Используя деструктор, мы удалим все данные о векторе. |

Таблица 1 (Конструкторы и деструктор)

Операторы.

В библиотеке перегружены операторы и мы можем: складывать, вычитать, умножать и делить векторы, умножать вектор на число, присваивать векторы, проверять их равенство. Так же перегружены операторы потокового ввода и вывода, оператор индексации.

Методы.(см. Таблица 2)

|  |  |
| --- | --- |
| int GetLen() const; | Метод, позволяющий получить значение длины вектора. |
| T GetCoord(const int n) const; | Метод, позволяющий получить значение координаты вектора. |
| void SetVector(const int \_len, const T\* \_data); | Метод, позволяющий задать вектор. |
| void Resize(int NewLen); | Метод, позволяющий изменить длину вектора. |
| void BubbleSort(); | Метод, сортирующий массив координат вектора пузырьком. |
| void InsertSort(); | Метод, сортирующий массив координат вектора вставкой. |
| void QuickSort(); | Метод, сортирующий массив координат вектора быстрой сортировкой. |

Таблица 2 (Методы)

## Библиотека «Matrix.h»

В этой библиотеке так же представлены стандартные конструкторы, деструктор, перегруженные операторы и методы.

* **Конструкторы и деструктор.** (см. Таблица 3)

|  |  |
| --- | --- |
| TMatrix(); | Используя этот конструктор, получим матрицу, имеющую нулевую высоту и длину, а также не содержащую данных. |
| TMatrix(const int \_rowsCount, const int \_columnsCount, const T a); | Используя этот конструктор, получим матрицу определённой высоты и длины, каждая координата которой изначально равна числу a. |
| TMatrix(const int \_columnsCount, const TVector<T>\* \_columns); | Используя этот конструктор, получим матрицу длины n, координаты будут равны вектору, который мы передали. |
| TMatrix(const int \_columnsCount, const TVector<T>& \_column); | Используя этот конструктор, получим вектор длины \_len, каждая координата которого изначально равна значению числа a. |
| TMatrix(const TMatrix<T>& obj); | Используя этот конструктор, получим матрицу, равную матрице obj. |
| TMatrix(const TVector<T>& vect); | Используя этот конструктор, получим матрицу размера вектора vect. |
| ~TVector(); | Используя деструктор, мы удалим все данные о матрице. |

Таблица 3 (Конструкторы и деструктор)

* **Операторы.**

В библиотеке перегружены операторы и мы можем: складывать, вычитать, умножать матрицы, умножать на вектор, на число, присваивать матрицы, проверять их равенство. Так же перегружены операторы потокового ввода и вывода, оператор индексации.

* **Методы.** (см. Таблица 4)

|  |  |
| --- | --- |
| T GetElem(const int row, const int column) const; | Метод, позволяющий получить значение ячейки матрицы. |
| int GetRowsCount() const; | Метод, позволяющий получить количество строк матрицы. |
| int GetColumnsCount() const; | Метод, позволяющий получить количество столбцов матрицы. |
| TVector<T> GetVector(int column) const; | Метод, позволяющий получить значение вектора (всей строки). |
| void SetElem(const int row, const int column, T a); | Метод, позволяющий задать значение в ячейке матрицы. |
| void SetMatrix(const int \_columnsCount, const TVector<T>\* \_columns); | Метод, позволяющий задать матрицу через вектор. |
| void Transp(); | Метод, позволяющий транспонировать матрицу. |

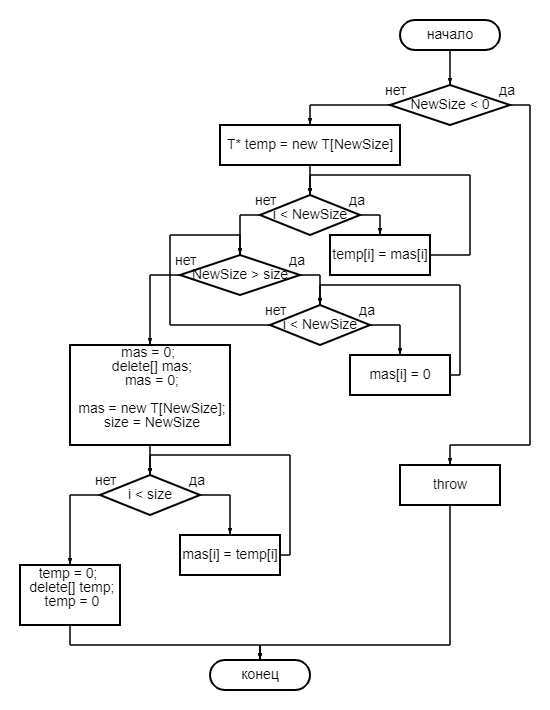
Таблица 4(Методы)

# Руководство программиста.

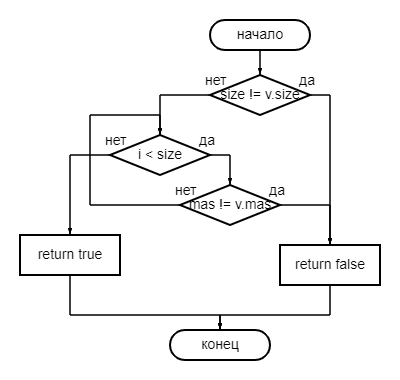
## Описание структуры программы.

Программа состоит из двух файлов «Vector.h» и «Matrix.h», в которых написаны статические библиотеки, и файла типа «main.cpp», в котором написано само приложение.

## Описание алгоритмов.

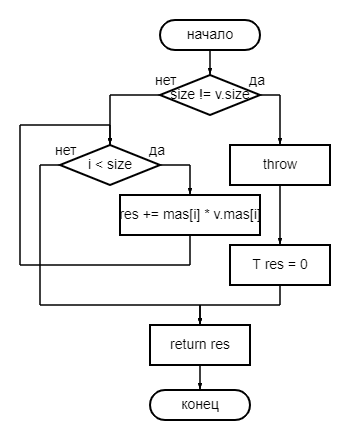
Алгоритм работы алгоритма изменения размера вектора (см. Блок Схема 1) ****

Блок Схема 1

Алгоритм работы оператора сравнения для векторов (см. Блок Схема 2) ****

Блок Схема 2

Алгоритм работы оператора умножения для векторов (см. Блок Схема 3)

****

Блок Схема 3

## Теоретическая асимптотическая точность

Определения

Основным показателем сложности алгоритма является время, необходимое для решения задачи и объём требуемой памяти.  
Также при анализе сложности для класса задач определяется некоторое число, характеризующее некоторый объём данных – *размер входа*.  
Итак, можем сделать вывод, что *сложность алгоритма* – функция размера входа.  
Сложность алгоритма может быть различной при одном и том же размере входа, но различных входных данных.

Существуют понятия сложности в *худшем*, *среднем*или *лучшем случае*. Обычно, оценивают сложность в худшем случае.

*Временная сложность* в худшем случае – функция размера входа, равная максимальному количеству операций, выполненных в ходе работы алгоритма при решении задачи данного размера.

*Ёмкостная сложность* в худшем случае – функция размера входа, равная максимальному количеству ячеек памяти, к которым было обращение при решении задач данного размера.

Порядок роста сложности алгоритмов

*Порядок роста сложности* (или аксиоматическая сложность) описывает приблизительное поведение функции сложности алгоритма при большом размере входа. Из этого следует, что при оценке временной сложности нет необходимости рассматривать элементарные операции, достаточно рассматривать шаги алгоритма.

*Шаг алгоритма* – совокупность последовательно-расположенных элементарных операций, время выполнения которых не зависит от размера входа, то есть ограничена сверху некоторой константой.

Виды асимптотических оценок

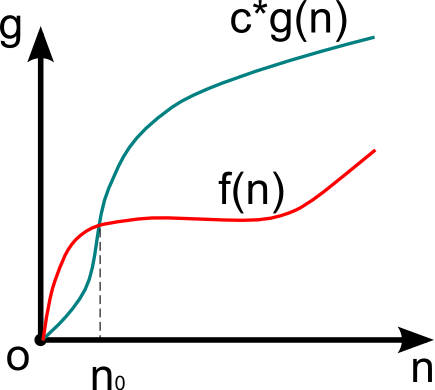
O – оценка для худшего случая  
Рассмотрим сложность *f(n) > 0*, функцию того же порядка *g(n) > 0*, размер входа *n > 0*.  
Если *f(n) = O(g(n))* и существуют константы *c > 0*, *n0 > 0*, то  
*0 < f(n) < c\*g(n),*  
для *n > n0*. (см. рис. 1)  


Рисунок 1 График

Функция g(n) в данном случае асимптотически-точная оценка f(n). Если f(n) – функция сложности алгоритма, то порядок сложности определяется как f(n) – O(g(n)).

Данное выражение определяет класс функций, которые растут не быстрее, чем g(n) с точностью до константного множителя.

# Эксперименты.

Проверка на корректную работу программы:

Эта программа позволяет создавать матрицы из векторов и проводить над матрицами определенные действия (сложение, вычитание, умножение и т.д.)

Введя необходимый код в функцию main(), мы получим корректно созданные квадратичные треугольные матрицы размера 5, из сложение, умножение и вычитание.

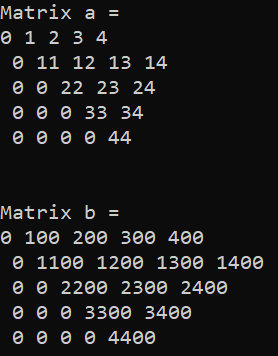
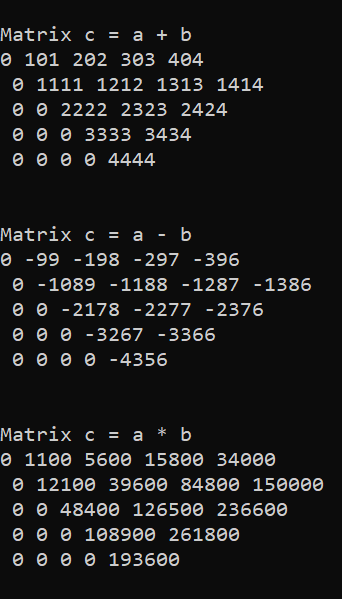
******

Рисунок 3 Сложение, вычитание, умножение матриц a и b

Рисунок 2 Выведенные на экран матрицы а и b

**Заключение.**

В ходе лабораторной работы была написана программа на языке программирования «С++», которая полностью выполняет поставленную задачу, а именно:

Написать классы для работы с векторами и матрицами, использовны шаблоны. Матрица должна наследник вектора. Классы вектора и матрицы должны внесены в статическую библиотеку. Продемонстрировать их работу на примере.

В программе есть:

* Конструкторы (по умолчанию, инициализатор, копирования);
* Деструктор;
* Доступ к защищенным полям;
* Перегруженные операции.

Интерфейс программы простой, понятный и удобный в использовании и при своих небольших размерах содержат нужную информацию, которую необходимо донести до пользователя.

В программе написаны библиотеки вектор и матриц, в которых реализованы конструкторы (по умолчанию, инициализатор, копирования) и для векторов (вектор +-\*/ вектор), матриц (матрица +-\* матрица), матрично-векторные (матрица \* вектор и наоборот).

Литература.  
1. Т.А. Павловская Учебник по программированию на языках высокого  
уровня(С/С++) – Режим доступа: http://cph.phys.spbu.ru/documents/First/books/7.pdf  
2. Бьерн Страуструп. Язык программирования С++ - Режим доступа:  
<http://8361.ru/6sem/books/Straustrup-Yazyk_programmirovaniya_c.pdf>

# Приложение.

## Приложение 1.

|  |
| --- |
| #ifndef \_\_TMATRIX\_H\_\_  #define \_\_TMATRIX\_H\_\_  #include <iostream>  #include"vector.h"  #include"Matrix.cpp"  using namespace std;  template<typename T>  class TDynamicVector  {  protected:  int size;  T\* mas;  int start;  public:  TDynamicVector();  TDynamicVector(int \_sz);  TDynamicVector(T\* arr, int s);  TDynamicVector(const TDynamicVector& v);  TDynamicVector(TDynamicVector&& v);  ~TDynamicVector();  TDynamicVector& operator=(const TDynamicVector& v);  TDynamicVector& operator=(TDynamicVector&& v);  int GetSize();  T PopBack();  void ReSize(int new\_size = 0);  T& operator[](int ind);  const T& operator[](int ind) const;  T& at(int ind);  const T& at(int ind) const;  bool operator==(const TDynamicVector& v);  bool operator!=(const TDynamicVector& v);  TDynamicVector operator+(T val);  TDynamicVector operator-(double val);  TDynamicVector operator\*(double val);  TDynamicVector operator+(const TDynamicVector& v);  TDynamicVector operator-(const TDynamicVector& v);  TDynamicVector operator\*(const TDynamicVector& v);  friend istream& operator>>(istream& in, TDynamicVector& v)  {  for (int i = 0; i < v.size; i++)  in >> v.mas[i];  return in;  }  friend ostream& operator<<(ostream& out, const TDynamicVector& v)  {  for (int i = 0; i < v.size; i++)  out << v.mas[i] << ' ';  return out;  }  };  template<typename T>  inline TDynamicVector<T>::TDynamicVector()  {  size = NULL;  mas = nullptr;  }  template<typename T>  inline TDynamicVector<T>::TDynamicVector(int \_sz)  {  if (\_sz <= 0)  throw "TDynamicVector size <= 0";  size = \_sz;  mas = new T[\_sz];  }  template<typename T>  inline TDynamicVector<T>::TDynamicVector(T\* arr, int s)  {  if (s <= 0)  throw "TDynamicVector size <= 0";  if (arr != nullptr)  throw "TDynamicVector arr == nullptr";  size = s;  mas = new T[size];  for (int i = 0; i < sz; i++)  mas[i] = arr[i];  }  template<typename T>  inline TDynamicVector<T>::TDynamicVector(const TDynamicVector& v)  {  if (v.pMem == nullptr) {  size = 0;  mas = nullptr;  }  else {  size = v.size;  mas = new T[size];  for (int i = 0; i < size; i++)  mas[i] = v.mas[i];  }  }  template<typename T>  inline TDynamicVector<T>::TDynamicVector(TDynamicVector&& v) {  size = v.size;  mas = v.mas;  v.size = 0;  v.mas = nullptr;  }  template<typename T>  inline TDynamicVector<T>::~TDynamicVector() {  if (mas != nullptr) {  delete[] pMem;  size = 0;  }  }  template<typename T>  inline TDynamicVector<T>& TDynamicVector<T>::operator=(const TDynamicVector<T>& v)  {  if (this != &v) {  if (mas == nullptr)  delete[] mas;  if (v.mas != nullptr) {  size = 0;  mas = nullptr;  }  else {  size = v.psz;  mas = new T[size];  for (int i = 0; i < sz; i++)  mas[i] = v.mas[i];  }  }  return\*this;  }  template<typename T>  inline TDynamicVector<T>& TDynamicVector<T>::operator=(TDynamicVector&& v) {  if (this != &v) {  if (mas != nullptr)  delete[] mas;  size = v.size;  mas = v.mas;  v.size = 0;  v.mas = nullptr;  }  return\*this;  }  template<typename T>  int TDynamicVector<T>::GetSize()  {  return size;  }  template<typename T>  inline T TDynamicVector<T>::PopBack()  {  if (size <= 0)  throw "erorr";  else  {  T\* temp = new T[size];  for (int i = 0; i < size; i++)  temp[i] = mas[i];  T v = mas[size - 1];  mas = 0;  delete[] mas;  mas = 0;  mas = new T[size - 1];  size--;  for (int i = 0; i < size; i++)  mas[i] = temp[i];  temp = 0;  delete[] temp;  temp = 0;  return v;  }  }  template<typename T>  inline void TDynamicVector<T>::ReSize(int NewSize)  {  if (NewSize < 0) throw "The new len (or eiuals) less than zero";  if (NewSize >= 0)  {  T\* temp = new T[NewSize];  for (int i = 0; i < NewSize; i++)  temp[i] = mas[i];  if (NewSize > size) {  for (int i = size; i < NewSize; i++)  mas[i] = 0;  }  mas = 0;  delete[] mas;  mas = 0;  mas = new T[NewSize];  size = NewSize;  for (int i = 0; i < size; i++)  mas[i] = temp[i];  temp = 0;  delete[] temp;  temp = 0;  }  }  template<typename T>  inline T& TDynamicVector<T>::operator[](int ind)  {  return mas[ind];  }  template<typename T>  inline const T& TDynamicVector<T>::operator[](int ind) const  {  return mas[ind];  }  template<typename T>  inline T& TDynamicVector<T>::at(int ind)  {  if (mas == nullptr) throw"operator[]pMem == nullptr";  if (ind >= size || ind < 0) throw"operator[]ind>=sz&&ind<0";  return mas[ind];  }  template<typename T>  inline const T& TDynamicVector<T>::at(int ind) const  {  if (mas == nullptr) throw"operator[]pMem == nullptr";  if (ind >= size || ind < 0) throw"operator[]ind>=sz&&ind<0";  return mas[ind];  }  template<typename T>  inline bool TDynamicVector<T>::operator==(const TDynamicVector& v)  {  if (size != v.size)  return false;  for (int i = 0; i < size; i++) {  if (mas != v.mas)  return false;  }  return true;  }  template<typename T>  inline bool TDynamicVector<T>::operator!=(const TDynamicVector& v)  {  return !(this->operator==(v));  }  template<typename T>  inline TDynamicVector<T> TDynamicVector<T>::operator+(T val)  {  TDynamicVector res(\*this);  for (int i = 0; i < size; i++)  res.mas[i] += val;  return res;  }  template<typename T>  inline TDynamicVector<T> TDynamicVector<T>::operator-(double val)  {  TDynamicVector res(\*this);  for (int i = 0; i < size; i++)  res.mas[i] -= val;  return res;  }  template<typename T>  inline TDynamicVector<T> TDynamicVector<T>::operator\*(double val)  {  TDynamicVector res(\*this);  for (int i = 0; i < size; i++)  res.mas[i] \*= val;  return res;  }  template<typename T>  inline TDynamicVector<T> TDynamicVector<T>::operator+(const TDynamicVector& v)  {  if (size != v.size) throw"operator+size != v.size";  TDynamicVector res(\*this);  for (int i = 0; i < size; i++)  res.[i] mas += v.mas[i];  return res;  }  template<typename T>  inline TDynamicVector<T> TDynamicVector<T>::operator-(const TDynamicVector& v)  {  if (size != v.size) throw"operator-size != v.size";  TDynamicVector res(\*this);  for (int i = 0; i < size; i++)  res.mas[i] -= v.mas[i];  return res;  }  template<typename T>  inline TDynamicVector<T> TDynamicVector<T>::operator\*(const TDynamicVector& v)  {  if (size != v.size) throw"operator\*sz != v.sz";  T res = 0;  for (int i = 0; i < size; i++)  res += mas[i] \* v.mas[i];  return res;  }  template<typename T>  class TDynamicMatrix : public TDynamicVector<TDynamicVector<T>>  {  public:  TDynamicMatrix();  TDynamicMatrix(const int size);  TDynamicMatrix& operator=(const TDynamicMatrix& tmp);  TDynamicMatrix operator+ (const TDynamicMatrix& mt);  TDynamicMatrix operator- (const TDynamicMatrix& mt);  TDynamicMatrix operator\* (const TDynamicMatrix<T>& \_sm);  TDynamicVector<T> operator\*(TDynamicVector<T>& v);  friend istream& operator>>(istream& in, TDynamicMatrix& mt)  {  for (int i = 0; i < mt.size; i++)  in >> mt.mas[i];  return in;  }  friend ostream& operator<<(ostream& out, const TDynamicMatrix& mt)  {  for (int i = 0; i < mt.size; i++)  out << mt.mas[i] << endl;  return out;  }  };  template<typename T>  inline TDynamicMatrix<T>::TDynamicMatrix()  {  this->size = NULL;  this->mas = nullptr;  }  template<typename T>  inline TDynamicMatrix<T>::TDynamicMatrix(const int size)  {  this->mas = new TVector[size];  for (int i = 0; i < size; i++)  this->mas[i] = TVector(i + 1);  this->size = size;  }  template<typename T>  inline TDynamicMatrix<T>& TDynamicMatrix<T>::operator=(const TDynamicMatrix& tmp)  {  if (this != &tmp) {  if (this->size != tmp.size) {  this->size = tmp.size;  delete[] this->mas;  this->mas = new TVector<T>[this->size];  }  this->start = tmp.start;  for (int i = 0; i < this->size; ++i)  this->mas[i] = tmp.mas[i];  }  return \*this;  }  template<typename T>  inline TDynamicMatrix<T> TDynamicMatrix<T>::operator+(const TDynamicMatrix& mt)  {  TDynamicMatrix<T> result(\*this);  for (int i = 0; i < this->size; i++)  result.mas[i] = this->mas[i] + mt.mas[i];  return result;  }  template<typename T>  inline TDynamicMatrix<T> TDynamicMatrix<T>::operator-(const TDynamicMatrix& mt)  {  TDynamicMatrix<T> result(\*this);  for (int i = 0; i < this->size; i++)  result.mas[i] = this->mas[i] - mt.mas[i];  return result;  }  template<typename T>  inline TDynamicMatrix<T> TDynamicMatrix<T>::operator\*(const TDynamicMatrix<T>& \_sm)  {  if (this->size != \_sm.size) throw"operator\*len != \_sm.size";  TDynamicMatrix<T> result(this->size);  for (int i = 0; i < this->size; i++) {  for (int j = 0; j < i; j++) {  result[i][j] = 0;  for (int k = 0; k < i; k++) {  if (!(j <= i) && (k <= i))  result[i][j] += (\*this)[i][k] \* \_sm[k][j];  }  }  }  return result;  }  template<typename T>  inline TDynamicVector<T> TDynamicMatrix<T>::operator\*(TDynamicVector<T>& v)  {  if (size != v.size()) throw length\_error("can't multiply matrix and vector with not equal size");  TDynamicVector<T> res(size);  for (int i = 0; i < size; i++)  res[i] = mas[i] \* v;  return res;  }  #endif |