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

Институт информационных технологий математики и механики

Отчёт по лабораторной работе

Динамические матрицы и вектора

Выполнил:

студент группы 3821Б1ФИ3

Мортина А.О.

Проверил:

заведующий лабораторией суперкомпьютерных технологий и высокопроизводительных вычислений

Лебедев И.Г.

Нижний Новгород

2022 г.

Содержание

[Введение 3](#_Toc122182292)

[1. Постановка задачи 4](#_Toc122182293)

[2. Руководство пользователя 5](#_Toc122182294)

3. Руководство программиста………………………………………………………………..7

[***а. Описание структуры кода программы*** 7](#_Toc122182295)

[***б. Описание структуры данных*** 8](#_Toc122182296)

[***в. Описание алгоритмов*** 8](#_Toc122182297)

[4. Эксперименты 14](#_Toc122182298)

[Заключение 15](#_Toc122182299)

[Список литературы 16](#_Toc122182300)

# Введение

Программирование – раздел информатики, изучающий методы и приёмы составления программ для компьютеров. Оно включает в себя разработку алгоритмов решения различных практических задач и их реализацию в виде компьютерных программ.

В узком смысле программированием называют процесс написания программы на конкретном языке программирования (этап кодирования).

Распределение по классам, есть везде и язык программирования С++ не является исключением. Классы являются основой С++. Эта структура данных позволяет группировать данные и методы внутри одной переменной.

В ходе выполнения лабораторный работы на языке программирования «С++» будет написана программа, в которой мы рассмотрим классы векторов и матриц, их методы, а также классическое применение механизма перегрузки стандартных операций. Такая программа позволит нам провести работу с объектами линейной алгебры.

# Постановка задачи

Нужно написать классы для работы с динамическими векторами и матрицами, использовав шаблоны. Матрица должна быть наследником вектора. Классы вектора и матрицы должны быть внесены в статическую библиотеку. В программе должны быть: конструкторы, деструктор, перегруженные операции: +,-,\*,/, =, ==,[ ], потоковый ввод и вывод.

# Руководство пользователя

Были созданы статические библиотеки (векторов и матриц), с помощью которой пользователь может работать с объектами (инициализировать, складывать и вычитать, умножать и делить, сравнивать, использовать ввод и вывод).

**Вектора**

1) Сложение и вычитание векторов. Из математики складываются/вычитаются попарные координаты векторов. (Сложение векторов (А+В). Результатом сложения векторов А и В называется вектор С=(а1+b1, a2+b2,…, an+bn)).

|  |
| --- |
| TVector<Type> operator + (const TVector<Type>& \_vector);  TVector<Type> operator - (const TVector<Type>& \_vector); |

2) Умножение вектора на вектор. Операция нахождения покоординатного нахождения двух векторов.

|  |
| --- |
| TVector<Type> operator \*(const TVector<Type>& \_vector) |

3) Деление вектора на вектор.

|  |
| --- |
| TVector<Type> operator / (const TVector<Type>& \_vector) |

4) Сравнение векторов. Операция позволяет попарно сравнить координаты векторов и определить, равны ли данные векторы или нет. Сравнение (А=В). Вектора считаются равными тогда и только тогда, когда ai=bi, при всех i=1..n.

|  |
| --- |
| bool operator == (const TVector<Type>& \_vector) |

**Матрицы**

1. Сложение и вычитание матриц. (Сложение матриц (А+В). Результатом сложения матриц А и В называется матрица С=(cij), где cij =( aij-bij), при всех i=1..m, j=1…n).

|  |
| --- |
| TMatrix<Type> operator + (const TMatrix<Type>& \_matrix)  TMatrix<Type> operator - (const TMatrix<Type>& \_matrix) |

1. Умножение матрицы на матрицу. (Умножение матриц (A\*В). Результатом умножения матриц A=(аij) и B=(bik) называется матрица С=(сik), где cij=aij\*bjk, при всех i=1…m, j=1…n, k=1…n).

|  |
| --- |
| TMatrix<Type> operator \* (const TMatrix<Type>& \_matrix) |

1. Умножение матрицы на вектор. Матрица умножается на вектор в соответствии с правилом «строк на столбец». При умножении матрицы на вектор столбцов количество столбцов в матрице должно соответствовать количеству строк в векторе столбцов.

|  |
| --- |
| TMatrix<Type> operator \* (const TVector<Type>& \_vector) |

1. Сравнение матриц. Операция позволяет попарно сравнить все соответствующие элементы двух матриц и определить, равны ли данные матрицы или нет. Сравнение (А==В). Матрицы считаются равными тогда и только тогда, когда aij=bij, при всех i=1..n, j=1…n.

|  |
| --- |
| bool operator == (const TMatrix<Type>& \_matrix) |

Потоковый ввод/вывод. Благодаря этим двум функциям мы можем либо вывести на экран значение необходимого нам вектора, матрицы или результат операции над этими двумя значениями (потоковый вывод) либо ввести в консоль значение вектора для других операций.

|  |
| --- |
| std::istream& operator>>(std::istream& stream, TVector<Type>& \_vector)  std::ostream& operator<<(std::ostream& stream, TVector<Type>& \_vector)  std::ostream& operator<<(std::ostream& stream, TMatrix<Type>& \_matrix)  std::istream& operator>>(std::istream& stream, TMatrix<Type>& \_matrix) |

**3. Руководство программиста**

## ***а. Описание структуры кода программы***

**Вектора**

1) Подключение библиотек:

|  |
| --- |
| #pragma once  #include <iostream> |

2) Объявление класса векторов и их защищенных полей:

|  |
| --- |
| T\*pMem; int size; |

3) Объявление в public конструкторов, деструктора, перегрузки операторов и методов:

|  |
| --- |
| public:  TDynamicVector();  TDynamicVector(size\_t \_size);  TDynamicVector(T\* arr, int s);  TDynamicVector(const TDynamicVector& v);  TDynamicVector(TDynamicVector&& v) noexcept;  ~TDynamicVector();  size\_t size() const noexcept { return sz };  TDynamicVector& operator=(const TDynamicVector& v);  TDynamicVector& operator=(TDynamicVector&& v) noexcept;  int GetSize();  T PopBack();  void ReSize(int NewSize = 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) const noexcept;  bool operator!=(const TDynamicVector& v) const noexcept;  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); |

**Матрицы**

1) Подключение библиотек:

|  |
| --- |
| #pragma once  #include <iostream>  #include"vector.h" |

5) Объявление класса матриц, как наследника векторов:

|  |
| --- |
| template<typename T>  class TDynamicMatrix : public TDynamicVector<TDynamicVector<T>> |

6) Объявление конструкторов, деструктора, перегрузки операторов и методов:

|  |
| --- |
| TDynamicMatrix();  TDynamicMatrix(const int size = 1);  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); |

## ***б. Описание структуры данных***

**Вектора**

1) Библиотеки: include <iostream>. #include <fstream>

2) Конструкторы: Vector( ), Vector(const TVector<Type>& \_vector)

3) Методы: GetSize

4) Функции: ~Tvector(), Resize

5) Операторы: operator [], operator =, operator +, operator -, operator \*, operator /

**Матрицы**

1) Библиотеки: include <iostream>

2) Конструкторы: TMatrix( ), TMatrix(int \_size)

3) Методы: Int GetSize()

4) Функции: ~TMatrix(), GetSize

5) Операторы: operator [], operator =, operator +, operator -, operator\*

## ***в. Описание алгоритмов***

**Вектора**

1. *Конструктор* по умолчанию задает начальные параметры защищенных полей объектов класса, приравнивая их к нулю.
2. *Деструктор* проверяет данные на то, чтобы они не были равны нулю, после чего, удаляет массив данных.
3. Оператор *сложения* покомпонентно в цикле складывает координаты векторов и записывает результат в созданный в данном блоке кода вектор result.

|  |
| --- |
| template<class T>  TVector<T> TVector<T>::operator + (const TVector<T>& \_vector)  {  if (\_vector.data == 0 && \_vector.length < 0) throw "empty variable";  if (\_vector.length != this->size) throw "error of length";  else{  TVector<T> result(\*this);  for (int q = 0; q < size; q++){  result.data[q] = (this->pMem[q]) + \_vector.data[q];  }  return result;  }} |

1. Оператор *вычитания* покомпонентно в цикле вычитает координаты векторов и записывает результат в созданный в данном блоке кода вектор result.

|  |
| --- |
| template<class T>  TVector<T> TVector<T>::operator-(const TVector<T>& \_vector)  {  if (\_vector.pMem == 0 && \_vector.size < 0) throw "empty variable";  if (\_vector.length != this->length) throw "error of length";  else{  TVector<T> result(\*this);  for (int q = 0; q < size; q++){  result.size[q] = (this->size[q]) - \_vector.size[q];  }  return result;  }} |

1. Оператор *умножения* реализует скалярное произведение векторов, которое равно сумме произведения соответствующих координат.

|  |
| --- |
| template<class T>  TVector<T> TVector<T>::operator/(const TVector<T>& \_vector)  {  if (\_vector.data == 0 && \_vector.size < 0) throw "empty variable";  if (\_vector.length != this->size) throw "size error";  else  {  TVector<T> result(\*this);  for (int q = 0; q < size; q++)  {  result.pMem[q] = (this->pMem[q]) \* \_vector.pMem[q];  }  return result;  }} |

1. *Деление* определено как обратная операция к умножению (скалярному произведению), т. е. – оно равно частному соответствующих координат.

|  |
| --- |
| template<class T>  TVector<T> TVector<T>::operator/(const TVector<T>& \_vector)  {  if (\_vector.pMem == 0 && \_vector.size < 0) throw "empty variable";  if (\_vector.size != this->size) throw "size error";  else{  TVector<T> result(\*this);  for (int q = 0; q < size; q++)  {  if (\_vector.pMem[q] != 0)  result. pMem [q] = (this- pMem ata[q]) / \_vector. pMem [q];  else throw "division by zero";  }return result;  }} |

1. Оператор *равенства* удаляет данные и создает новый массив для хранения данных. После чего, покомпонентно, в цикле записывает в данный массив данные из входного вектора.

|  |
| --- |
| template<class T>  inline TVector<T>& TVector<T>::operator=(const TVector<T>& \_vector)  {  if (data == 0) data = new T[\_vector.length];  else if (data != 0)  {  data = 0;  delete[] data;  data = 0;  data = new T[\_vector.length];  }  length = \_vector.length;  for (int q = 0; q < length; q++)  {  data[q] = \_vector.data[q];  }  return \*this;  } |

1. Оператор *сравнения* сравнивает с помощью оператора if сначала самокопирование, затем длины, а позже в цикле for также покомпонентно сравнивает данные на равенство.

|  |
| --- |
| template<class T>  inline bool TVector<T>::operator==(const TVector<T>& \_vector)  {  if (length != \_vector.length) return false;  for (int q = 0; q < length; q++)  {  if (data[q] != \_vector.data[q]) return false;  }return true;  } |

**Матрицы**

1. *Конструктор* по умолчанию (ничего не получает на вход) приравнивает к нулю все защищенные поля класса матрицы.
2. *Деструктор* проверяет с помощью функции if данные на их неравенство нулю, после чего данные удаляются и дате присваивается значение нуля.
3. Перегруженные операторы *ввода и вывода* реализованы схожем образом с их реализацией в классе векторов (в данном классе вместо одного цикла теперь два).
4. Оператор *присваивания* сначала проводит проверки на присваивание самого себя, после чего покомпонентно копирует данные из входной матрицы в исходную.

|  |
| --- |
| template<class T>  inline TMatrix<T>& TMatrix<T>::operator=(const TMatrix<T>& \_matrix){  if (this == &\_matrix) return \*this;  if (this->length != \_matrix.length || width != \_matrix.width) throw "error - lenght's of matrix not equals";  for (int q = 0; q < width; q++)  for (int w = 0; w < this->length; w++)  data[q][w] = \_matrix.data[q][w];  return \*this;  } |

1. Операторы *сложения и вычитания* реализованы с точностью до знаков + - у компонент при действии с ними. Для записи результата действия создается объект класса матриц result, после чего, двойным циклом складываем ил вычитаем покомпонентно данные матриц.

|  |
| --- |
| template<class T>  inline TMatrix<T> TMatrix<T>::operator+(const TMatrix& \_matrix){  if (this->length != \_matrix.length || width != \_matrix.width) throw "error - lenght's of matrix not equals";  TMatrix result(\*this);  for (int q = 0; q < width; q++)  for (int w = 0; w < this->length; w++)  result.data[q][w] = this->data[q][w] + \_matrix.data[q][w];  return result;  }  template<class T>  inline TMatrix<T> TMatrix<T>::operator-(const TMatrix<T>& \_matrix){  if (this->length != \_matrix.length || width != \_matrix.width) throw "error - lenght's of matrix not equals";  TMatrix result(\*this);  for (int q = 0; q < width; q++)  for (int w = 0; w < this->length; w++)  result.data[q][w] = this->data[q][w] - \_matrix.data[q][w];  return result;  } |

1. Оператор *умножения матриц* реализован с созданием результирующей матрицы result, после чего в двойном цикле для каждой ячейки результирующей матрицы производим сложение произведений соответствующих данных по I’той строке из 1й матрицы для соответствующих данных из j’того столбца 2й матрицы.
2. Оператор *умножения матрицы на вектор* реализован с алгебраической точки зрения также, как и оператор умножения матриц, но в котором мы умножаем на один столбец.

|  |
| --- |
| template<class T>  TMatrix<T> TMatrix<T>::operator\*(const TVector<T>& \_vector){  TVector<T> temp(\_vector);  if ((this->length) != temp.GetLength()) throw "error - parametrs not compared";  TMatrix<T> result(1, width, 0);  for (int i = 0; i < width; i++){  for (int j = 0; j < 1; j++){  result[i][j] = 0;  for (int x = 0; x < this->length; x++)  result[i][j] += this->data[i][x] \* temp[x];  }}  result.Print();  return result;  } |

**Асимптотическая сложность** - рассмотрение входных данных большого размера и оценка порядка роста времени работы алгоритма. Обычно алгоритм с меньшей асимптотической сложностью является более эффективным для всех входных данных, за исключением лишь, возможно, данных малого размера.

Сложность алгоритма может быть различной при одном и том же размере входа, но различных входных данных.

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*.  
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.

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.

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.

# Эксперименты

Проверка на корректность работы программы:

Программа даёт возможность создания матриц из векторов, а также проведения над матрицами алгебраических действий: сложения, вычитания, умножения и тд.

После ввода кода и данных в функцию, получаем правильно созданные треугольные матрицы, а также их результаты алгебраических действий при работе с ними.

1. Ввод на экран двух матриц «a» и «b».(Рис.1).

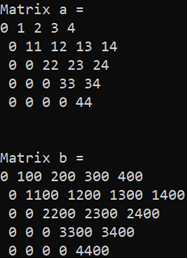
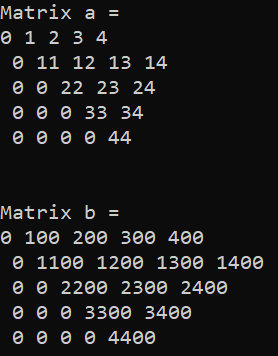
******

Рис.1

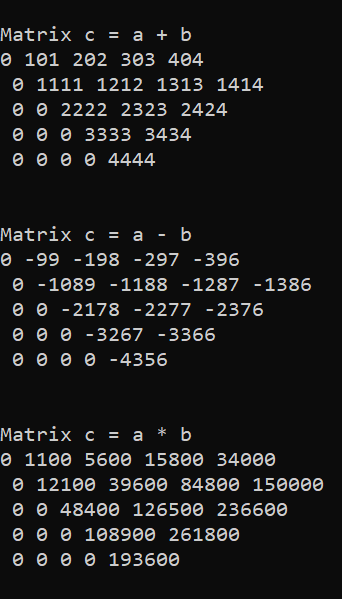
1. Данные и результат сложения(Рис.2), вычитания(Рис.3) и умножения двух матриц.(Рис.4).

Рис2.

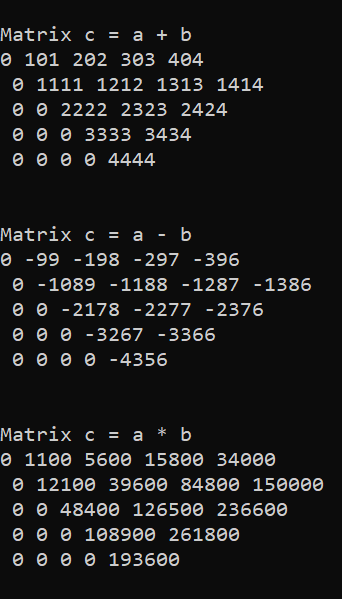


Рис.3

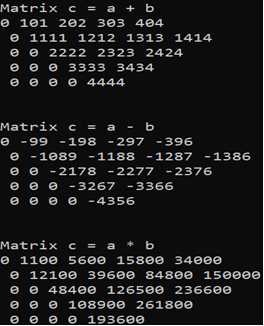


Рис.4

Из-за некорректности различного ввода данных программа выдавала неверные результаты.

# Заключение

В ходе лабораторной работы мною была написана программа на языке «С++», которая удовлетворяет поставленной задаче. Написаны классы для работы с векторами и матрицами, использованы шаблоны. Классы вектора и матрицы внесены в статическую библиотеку. Конструктор, деструктор и перегруженные операции имеются. Работа продемонстрирована на примере.

Выполнение данной лабораторной работы позволило мне лучше изучить и понять принципы работы с языком «С++», а именно работы с написанием классов для векторов и матриц, при этом используя шаблон.

Таким образом, выполнение такой лабораторной работы позволило мне значительно улучшить свои навыки программирования, больше разобраться с языком программирования, и, конечно же, применить и отработать все полученные знания на практике.

# Список литературы

1. Лекции Сысоева Александра Владимировича, 2022.
2. Практики и лабораторные Лебедева Ильи Геннадьевича, 2022.
3. Рацеев, С. М. Программирование на языке Си : учебное пособие для вузов / С. М. Рацеев. — Санкт-Петербург : Лань, 2022. — 332 с. — ISBN 978-5-8114-8585-7. — Текст : электронный // Лань : электронно-библиотечная система.
4. Т.А. Павловская Учебник по программированию на языках высокого  
   уровня(С/С++) – Режим доступа: <http://cph.phys.spbu.ru/documents/First/books/7.pdf>
5. Бьерн Страуструп. Язык программирования С++ - Режим доступа:  
   <http://8361.ru/6sem/books/Straustrup-Yazyk_programmirovaniya_c.pdf>