МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное автономное образовательное учреждение

высшего образования

**«Национальный исследовательский**

**Нижегородский государственный университет им. Н.И. Лобачевского»**

**(ННГУ)**

Отчёт по лабораторной работе

**Векторы**

Выполнил:

студент группы 3821Б1ПМ3

Еремеев В.П.

Проверил:

Заведующий лабораторией суперкомпьютерных технологий и высокопроизводительных вычислений

Лебедев И.Г.

Нижний Новгород

2022 г.

Содержание

[Введение 3](#_Toc104067892)

[1. Постановка задачи 4](#_Toc104067893)

[2. Руководство пользователя 5](#_Toc104067894)

[3. Руководство программиста 10](#_Toc104067895)

[a. Описание структуры программы 10](#_Toc104067896)

[b. Описание структур данных 15](#_Toc104067897)

[c. Описание алгоритмов 18](#_Toc104067898)

[4. Эксперименты 25](#_Toc104067899)

[Заключение 26](#_Toc104067900)

[Литература 27](#_Toc104067901)

# Введение

Весь реальный мир состоит из объектов. Города состоят из районов, в каждом районе есть свои названия улиц, на каждой улице находятся жилые дома, которые также состоят из объектов. Практически любой материальный предмет можно представить в виде совокупности объектов, из которых он состоит.

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 данной лабораторной работе мы рассмотрим классы векторов и матриц, их методы, а так же классическое применение механизма перегрузки стандартных операций. Данная программа позволит нам провести работу с объектами линейной алгебры.

# Постановка задачи

Необходимо написать классы для работы с векторами и матрицами, использовав шаблоны. Матрица должна быть наследником вектора. Классы вектора и матрицы должны быть внесены в статическую библиотеку. Необходимо продемонстрировать их работу на примере.

В программе 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), матриц(матрица +,-,\* матрица), матрично-векторные(матрица \* вектор и наоборот).

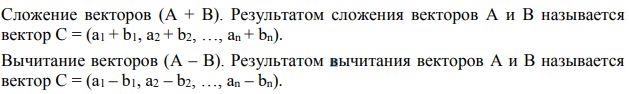
# Руководство пользователя

Была написана библиотека векторов и матриц, в которой пользователь может работать с данными объектами, а именно задавать, удалять, инициализировать, складывать, вычитать, умножать, делить, сравнивать, присваивать, а так же производить потоковый ввод и вывод.

***Сложение и вычитание векторов***

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

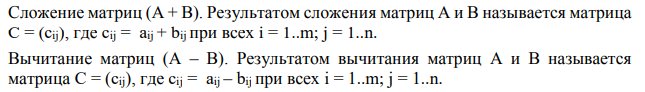
С точки зрения математики, складывает(вычитает) попарно координаты двух векторов. (аналогично складываются матрицы)



*Рисунок 1 Сложение и вычитание векторов в математике*

***Сложение и вычитание матриц***

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



*Рисунок 2 Сложение и вычитание матриц в математике*

***Умножение вектора на вектор***

* TVector<Type> operator \*(const TVector<Type>& \_vector);
* Type ScalarMult(const TVector<Type>& FirstVector, const TVector<Type>& SecondVector);

Данная операция позволяет найти покоординатное произведение двух векторов.

***Умножение матрицы на вектор***

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

****

*Рисунок 3. Умножение матрицы на вектор в математике*

Эта операция позволяется перемножать матрицу и вектор. При умножении матрицы на вектор, надо рассматривать вектор как вектор-столбец. 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.

***Умножение матрицы на матрицу***

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



*Рисунок 4 Умножение матрицы на матрицу в математике*

***Деление вектора на вектор***

* TVector<Type> operator / (const TVector<Type>& \_vector);

Данная операция позволяет найти покоординатное деление двух векторов

***Равенство двух векторов***

* bool operator == (const TVector<Type>& \_vector);



*Рисунок 5 Сравнение векторов в математике*

Данная операция позволяет попарно сравнить координаты векторов и определить, равны ли данные векторы или нет.

Данный оператор возвращает вектор. Возвращается True, если векторы равны, то есть равны числа, которые в них хранятся или же если совпадают адреса сравниваемых объектов.

Возвращается False, если числа, хранящиеся внутри векторов не равны.

***Сравнение двух матриц***

* bool operator == (const TMatrix<Type>& \_matrix);



*Рисунок 6 Сравнение матриц в математике*

Данная операция позволяет попарно сравнить все соответствующие элементы двух матриц и определить, равны ли данные матрицы или нет.

Данный оператор возвращает вектор. Возвращается True, если матрицы равны, то есть равны числа, которые в них хранятся, или же если совпадают адреса сравниваемых объектов. Возвращается False, если числа, хранящиеся внутри матриц не равны.

***Потоковый ввод и вывод***

Векторов:

* std::istream& operator>>(std::istream& stream, TVector<Type>& \_vector)
* std::ostream& operator<<(std::ostream& stream, TVector<Type>& \_vector)

Матриц:

* std::ostream& operator<<(std::ostream& stream, TMatrix<Type>& \_matrix)
* std::istream& operator>>(std::istream& stream, TMatrix<Type>& \_matrix)

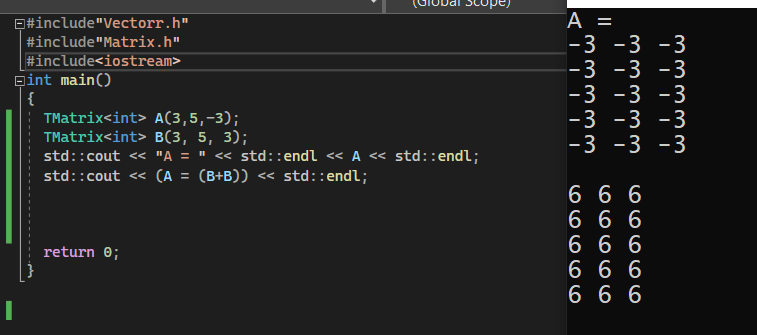
Благодаря этим двум функциям мы можем либо вывести на экран значение необходимого нам вектора, матрицы или результат операции над этими двумя значениями (потоковый вывод) либо ввести в консоль значение вектора для других операций. Данный оператор возвращает поток ostream/istream, которые используются для cout и cin.

***Оператор индексации []***

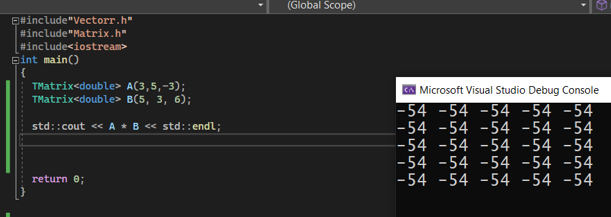
С помощью данного оператора мы можем получить адрес конкретной ячейки данного вектора и в дальнейшем работать с ней как с простым числом, можем получить его значение. Благодаря этому оператору мы можем работать с объектами класса Vector, как с обычными массивами, с которыми привыкли работать.

С помощью данного оператора мы можем получить адрес конкретной ячейки данной матрицы и в дальнейшем работать с ней как с простым числом, можем получить его значение. Благодаря этому оператору мы можем работать с объектами класса Matrix, как с обычными массивами, с которыми привыкли работать.

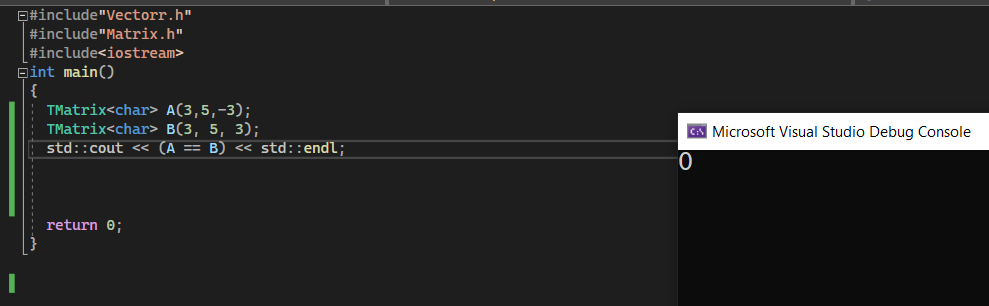
***Примеры вывода:***



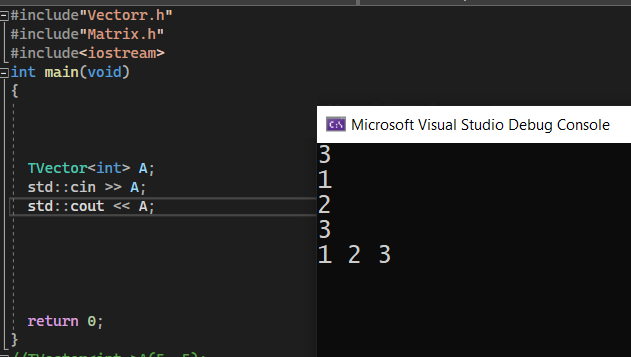
*Рисунок 7 Пример арифметических действий с матрицами*



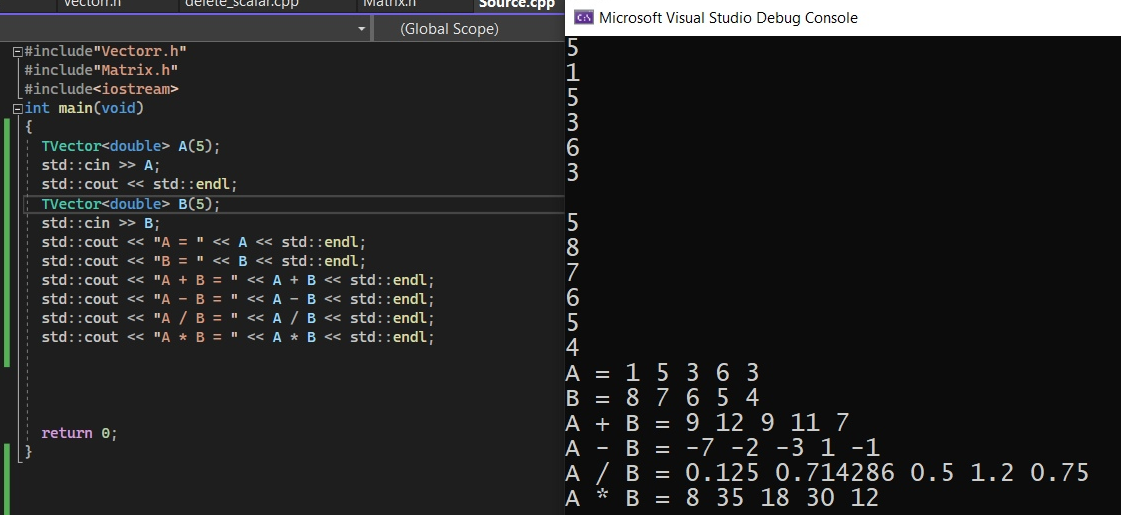
*Рисунок 8 Пример умножение матриц*



*Рисунок 9 Пример сравнения матриц*



*Рисунок 10 Потоковый ввод и вывод векторов*



*Рисунок 11 Пример бинарных математических операций с векторами*

# Руководство программиста

## Описание структуры программы

***Векторы***

Подключение необходимых библиотек.

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

Фрагмент кода 1

Объявление класса векторов и их защищенных полей (длина и массив данных). Для инициализации используем шаблон класса. (массив T\* data – массив для хранения данных объекта, поле length отвечает за хранение длины вектора).

|  |
| --- |
| protected:  Type\* data;  int length; |

Фрагмент кода 2

Объявляем в секции public конструктор по умолчанию, деструктор, конструктор копирования, методы класса и перегрузку операторов.

|  |
| --- |
| public:    TVector(int \_length = 0);  TVector(const TVector<Type>& \_vector);  ~TVector();  Type GetLength();  Type PopBack();  void PushBack(Type variable);  Type& operator[](int number);  TVector<Type> operator + (const TVector<Type>& \_vector);  TVector<Type> operator - (const TVector<Type>& \_vector);  TVector<Type> operator / (const TVector<Type>& \_vector);  TVector<Type>& operator = (const TVector<Type>& \_vector);  TVector<Type> operator \*(const TVector<Type>& \_vector);  Type ScalarMult(const TVector<Type>& FirstVector, const TVector<Type>& SecondVector);  void ReSize(int NewLength = 0);  bool operator == (const TVector<Type>& \_vector); |

Фрагмент кода 3

Реализуем перегрузку потоковых операторов ввода и вывода для класса векторов (операторы подробно освещаются в описании алгоритмов) .

Реализуем конструктор и деструктор

|  |
| --- |
| template<class Type>  inline TVector<Type>::TVector(int \_length)  {  data = 0;  data = new Type[\_length];  length = \_length;  }  template<class Type>  inline TVector<Type>::~TVector()  {  if (data != 0) delete[] data;  data = 0;  } |

Фрагмент кода 4

Реализация конструктора копирования.

|  |
| --- |
| template <class Type>  inline TVector<Type>::TVector(const TVector<Type>& \_vector)  {  if (\_vector.data == 0) throw "varriable of vector is empty!";  else  {  if (data != 0)  {  data = 0;  delete[] data;  data = 0;  data = new Type[\_vector.length];  }  else  data = new Type[\_vector.length];  length = \_vector.length;  for (int q = 0; q < length; q++)  {  data[q] = \_vector.data[q];  }  }  } |

Фрагмент кода 5

Реализация метода дающего доступ к защищенному полю длины вектора.

|  |
| --- |
| template<class Type>  inline Type TVector<Type>::GetLength()  {  return length;  } |

Фрагмент кода 6

Реализация всех методов класса векторов. В данном блоке объеденины все перегрузки операторов, имеющиеся в данном классе, в них входят: оператор сложения, вычитания, умножения, деления, обращения к конкретной координате, присвоения, равенства.

Подробнее мы их рассмотрим в разделе «описания алгоритмов».

Реализация метода, позволяющего изменить длину исходного вектора.

|  |
| --- |
| template<class Type>  inline void TVector<Type>::ReSize(int NewLength)  {  if (NewLength < 0) throw "The new length (or equals) less than zero";  if (NewLength >= 0)  {  Type\* temporary = new Type[NewLength];  for (int q = 0; q < NewLength; q++)  {  temporary[q] = data[q];  }  data = 0;  delete[] data;  data = 0;  data = new Type[NewLength];  length = NewLength;  for (int q = 0; q < length; q++)  data[q] = temporary[q];  temporary = 0;  delete[] temporary;  temporary = 0;  }  } |

Фрагмент кода 7

***Матрицы***

Подключение библиотек.

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

Фрагмент кода 8

Объявление класса матриц как публичного наследника класса векторов. Также в данном блоке в классе матриц добавляется защищенное поле (ширина матрицы).

|  |
| --- |
| template <class Type>  class TMatrix : public TVector<TVector<Type>>  {  protected:    int width; |

Фрагмент кода 9

Объявления всех методов класса, конструкторов (по умолчанию, создания, копирования), деструктора, перегрузки операторов.

|  |
| --- |
| public:  TMatrix();  ~TMatrix();  TMatrix(int \_length);  TMatrix(int \_length, int width, Type variable);  TMatrix(const TMatrix<Type>& \_matrix);  int GetWidth();  int GetLength();  TMatrix<Type> operator + (const TMatrix<Type>& \_matrix);  TMatrix<Type> operator - (const TMatrix<Type>& \_matrix);  TMatrix<Type>& operator = (const TMatrix<Type>& \_matrix);  TMatrix<Type> operator \* (const TMatrix<Type>& \_matrix);  TMatrix<Type> operator \* (const TVector<Type>& \_vector);  bool operator == (const TMatrix<Type>& \_matrix);  void Show(); |

Фрагмент кода 10

Реализуем перегруженные методы ввода и вывода для класса матриц.

Реализация конструктора по умолчанию конструктора и деструктора

|  |
| --- |
| template<class Type>  inline TMatrix<Type>::TMatrix()  {  this->length = 0;  this->width = 0;  this->data = 0;  }  template<class Type>  inline TMatrix<Type>::~TMatrix()  {  if (data != 0)  {  for (int q = 0; q < width; q++)  {  this->data[q].~TVector();  this->data[q] = 0;  }    this->data = 0;  delete[] this->data;  this->data = 0;  }  } |

Фрагмент кода 11

Реализация конструктора копирования.

|  |
| --- |
| template<class Type>  inline TMatrix<Type>::TMatrix(const TMatrix<Type>& \_matrix)  {  width = \_matrix.width;  this->length = \_matrix.length;  this->data = new TVector<Type>[width];  for (int q = 0; q < width; q++)  this->data[q] = TVector<Type>(this->length);  this->length = \_matrix.length;  for (int q = 0; q < \_matrix.width; q++)  for (int w = 0; w < \_matrix.length; w++)  this->data[q][w] = \_matrix.data[q][w];  } |

Фрагмент кода 12

Реализация методов, позволяющих получить защищенные данные объекта класса.

|  |
| --- |
| template<class Type>  inline int TMatrix<Type>::GetWidth()  {  return width;  }  template<class Type>  inline int TMatrix<Type>::GetLength()  {  return this->length;  } |

Фрагмент кода 13

Реализация перегрузки операторов данного класса, а именно: оператор сложения, вычитания, умножения матриц, умножения матрицы на вектор, деления, оператор присвоения и равенства.

Также как и с векторами, операторы более подробно освещается в реализации «описания алгоритмов»

## Описание структур данных

**Векторы**

Class TVector является программной реализацией геометрического вектора. В классе описаны математические операции, применимые к векторам. Класс вектора стандартной библиотеки C++ — это шаблон класса, представляющий реализацию контейнера. Вектор хранит элементы заданного типа в линейном расположении.

Библиотеки:

«#include <iostream>» – Объявляет объекты, управляющие чтением из стандартных потоков и записью в них;

«#include <fstream>» – используется для перегрузки операторов потокового ввода и вывода - даёт возможности для работы с потоками.

Конструкторы:

«Vector( )» – Этот конструктор создает стандартный вектор;

Vector(int size)» – Этот конструктор принимает аргумент, равный требуемому размеру, и создает вектор, начальная длина которого определяется size;

«Vector(const TVector<Type>& \_vector)» – конструктор копирования, который принимает на вход константную(для защиты данных) ссылку на TVector шаблонного типа Type.

Типы: **«**Type» – представляет шаблонный тип данных, хранящихся в векторе.

Поля класса:

«Type\* data» – динамический массив данных, в котором хранятся переменный типа Type;

«int length» – длина нашего вектора.

Метода класса:

«GetLength» – Возвращает количество элементов в векторе;

«pop\_back» – Удаляет элемент в конце вектора;

«push\_back» – Добавляет элемент в конец вектора.

Функции:

«~Tvector()» – это деструктор, который вызывается автоматически при выходе объекта из области действия или явно уничтожена вызовом delete;

«Resize» – Определяет новый размер вектора.

Операторы:

«operator []» – Возвращает ссылку на элемент вектора в указанной позиции;

«operator =» – Заменяет элементы вектора копией другого вектора;

«operator +» – Поэлементное сложение векторов;

«operator -» – Поэлементное вычитание векторов;

«operator \*» – Поэлементное умножение векторов;

«operator /» – Поэлементное деление векторов.

**Матрицы**

Класс TMatrix содержит три поля для хранения числа строк, столбцов и элементов матрицы в виде двумерного динамического массива, представленного шаблонным классом TVector.

Библиотеки:

#include <iostream> – Объявляет объекты, управляющие чтением из стандартных потоков и записью в них.

Конструкторы:

«TMatrix( )» – Этот конструктор создает матрицу;

«TMatrix(int \_lenght)» – Этот конструктор принимает аргумент, равный требуемому размеру, и создает матрицу, начальная емкость которой определяется length;

«TMatrix(int lenght, int width, int incr)» – Этот конструктор создает матрицу, чья начальная емкость задается «lenght» и «width» .

Поля класса:

«Type\* data» – динамический массив данных, в котором хранятся переменный типа Type, причём Type\* data наслледуется от класса TVector;

«int length» – длина нашей матрицы наслледуется от класса TVector;

«int width» – ширина нашей матрицы;

Методы класса:

«int GetWidth()» – возвращает количество столбцов;

«Int GetLength()» – возвращает количество строк.

Функции:

«~TMatrix()» – это деструктор, который вызывается автоматически при выходе объекта из области действия или явно уничтожена вызовом delete;

«GetWidth» – возвращает ширину матрицы;

«GetLength» – возвращает длину матрицы.

Операторы:

«operator []»– Возвращает ссылку на элемент матрицы в указанной позиции;

«operator =» – Заменяет элементы матрицы копией другой матрицы;

«operator+» – Поэлементное сложение матриц;

«operator -» – Поэлементное вычитание матриц;

«operator \*» – Умножение матриц ( матрица\* матрица / матрица\*вектор).

## Описание алгоритмов

**Векторы:**

Перегруженные операторы ввода и вывода работают по схожему алгоритму: в цикле for (итерации – от 1 до длины исходного объекта класса) мы записываем или выводим информацию, в зависимости от оператора.

|  |
| --- |
| template<class T>  std::istream& operator>>(std::istream& stream, TVector<T>& \_vector)  {  for (int i = 0; i < \_vector.GetLength(); i++)  stream >> \_vector[i];  return stream;  }  template<class T>  std::ostream& operator<<(std::ostream& stream, TVector<T>& \_vector)  {  for (int q = 0; q < \_vector.GetLength(); q++)  {  stream << \_vector[q] << ' ';  }  return stream;  } |

Фрагмент кода 14

Конструктор по умолчанию задает начальные параметры защищенных полей объектов класса, приравнивая их к нулю. (входные данные не требуются).

Объявление функции для вывода массива на экран.

Конструкторы создания объектов требуют некоторые входные данные: один – длину вектора, второй – длину вектора и данные о координате. Первый создает объект указанной пользователем длины, заполненный нулями, второй же также создает вектор заданной длины, но уже с указанной пользователем датой.

Конструктор копирования сначала проверяет данные, на то, пустые ли они, после чего создает массив данных, длина которого равна длине заданного объекта, после чего в цикле записывает в данный массив данные из исходного вектора.

Деструктор проверяет данные на то, чтобы они не были равны нулю, после чего, удаляет массив данных.

Метод GetLength() возвращает длину объекта, который его вызвал.

Далее во всех алгебраических методах встречается «проверка на непутстоту» данных и равенство длин векторов, реализованных через «if».

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->length) throw "error of length";  else  {  TVector<T> result(\*this);  for (int q = 0; q < length; q++)  {  result.data[q] = (this->data[q]) + \_vector.data[q];  }  return result;  }  } |

Фрагмент кода 15

Оператор вычитания покомпонентно в цикле вычитает (исходный вектор – уменьшаемое, подающийся на вход - вычитаемое) 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->length) throw "error of length";  else  {  TVector<T> result(\*this);  for (int q = 0; q < length; q++)  {  result.data[q] = (this->data[q]) - \_vector.data[q];  }  return result;  }  } |

Фрагмент кода 16

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.length < 0) throw "empty variable";  if (\_vector.length != this->length) throw "lenght error";  else  {  TVector<T> result(\*this);  for (int q = 0; q < length; q++)  {  result.data[q] = (this->data[q]) \* \_vector.data[q];  }  return result;  }  } |

Фрагмент кода 17

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.data == 0 && \_vector.length < 0) throw "empty variable";  if (\_vector.length != this->length) throw "lenght error";  else  {  TVector<T> result(\*this);  for (int q = 0; q < length; q++)  {  if (\_vector.data[q] != 0)  result.data[q] = (this->data[q]) / \_vector.data[q];  else throw "division by zero";  }  return result;  }  } |

Фрагмент кода 18

Оператор равенства удаляет данные и создает новый массив для хранения данных (длина берется у входного вектора). 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;  } |

Фрагмент кода 19

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;  } |

Фрагмент кода 20

Метод ReSize() на входе проверяет неотрицательна ли новая длина, после чего создает массив для временного хранения данных. После чего старые данные из исходного массива удаляются, его длина меняется на пользовательскую (введеную ранее), и в цикле из временного массива данных в массив данных нового вектора переносятся значения, после чего временное хранилище удаляется.

**Матрицы:**

Стоит сказать о том, что матрицы как класс реализованы как наследник от класса векторов, шаблонным классом которого является сам класс векторов, потому в защищенных полях добавляется только ширина матрицы. (из класса векторов копируются поля T\* data и length).

Стоит заметить, что к данным (дате) в данном классе нужно обращаться через указатель this-> т. к данный класс является наследником от векторов.

Перегруженные операторы ввода и вывода реализованы схожем образом с их реализацией в классе векторов, только в данном классе вместо одного цикла теперь два(ходим по каждой строке, фиксируя ее, после чего, для фиксированной строки начинаем проходить по столбцам и выводить данные или же записывать их).

|  |
| --- |
| template<class T>  std::ostream& operator<<(std::ostream& stream, TMatrix<T>& \_matrix)  {  for (int q = 0; q < \_matrix.GetWidth(); q++)  {  for (int i = 0; i < \_matrix.GetLength(); i++)  stream << \_matrix[q][i] << ' ';  stream << std::endl;  }  return stream;  }  template<class T>  std::istream& operator>>(std::istream& stream, TMatrix<T>& \_matrix)  {  T var = 0;  for (int q = 0; q < \_matrix.GetWidth(); q++)  {  for (int i = 0; i < \_matrix.GetLength(); i++)  {  stream >> var;  \_matrix[q][i] = var;  }  }  return stream;  } |

Фрагмент кода 21

Конструктор по умолчанию (ничего не получает на вход) приравнивает к нулю все защищенные поля класса матрицы.

Конструкторы создания (первый получает на вход длину матрицы, другой – длину, ширину и шаблонный параметр данных) реализованы схожим образом. Первый создает матрицу заданной ширины, длина задана пользователем, данные равны нулю. Второй же длину и ширину получает из входных данных пользователя, так же как и шаблонную единицу для заполнения данных.

Конструктор копирования записывает в поля данной матрицы ширину, а также создает новый массив из векторов размера ширины матрицы. После чего каждый вектор в цикле копируется в новый вектор векторов. Последним шагом в двойном цикле данные из входной матрицы записываются в данную (исходную).

Деструктор проверяет с помощью функции if данные на их неравенство нулю, после чего данные удаляются и дате присваивается значение нуля.

Методы GetLength() и GetWidth() позволяют пользователю получить доступ к защищенным полям класса матрицы через return… (обращаемся к полям также через this->).

Также, как и в векторах у всех операторов сначала проходят проверки на размеры матриц: для сложения и вычитания – равенство размеров, для умножения – равенство смежных размеров (ширины одной и длины второй)

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 (построчно, фиксируя строку, и по столбцам) 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;  } |

Фрагмент кода 22

Оператор присваивания сначала проводит проверки на присваивание самого себя, после чего покомпонентно копирует данные из входной матрицы в исходную (также там проводится проверка на соответствие размеров).

|  |
| --- |
| 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;  } |

Фрагмент кода 23

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.

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;  } |

Фрагмент кода 24

Оператор равенства сравнивает параметры длины и ширины, адреса сравниваемых объектов, после чего покомпонентно сравниваются элементы матрицы.

# Эксперименты

После написания программы, ее необходимо протестировать при разных исходных данных, чтобы оценить время работы программы и общую работоспособность. Ниже будет приведена таблица со временем работы программы при разном размере входных данных.

Замерим время работы программы для некоторых арифметических действий. Для этого воспользуемся библиотекой «time.h».

Для замеров воспользуемся командами:

clock\_t start = clock();

clock\_t end = clock();

double seconds = (double)(end - start) / CLOCKS\_PER\_SEC;

std::cout << "seconds = " << seconds << std::endl;

|  |  |  |
| --- | --- | --- |
| **Операция** | **Размер входных данных** | **Приблизительное время работы, мс** |
| Умножения матриц  1 замер | 100x100 | 0,003 |
| Умножения матриц  2 замер | 1000x1000 | 3,988 |
| Сложение матриц 1 замер | 100x100 | 0,002 |
| Сложение матриц 2 замер | 1000x1000 | 0,250 |

Сложность алгоритма умножения матрицу на матрицу o(n^3).

Сложность алгоритма сложения и вычитания матрицы на матрицу o(n^3).

Так как сложность алгоритма сложения равна o(n^2), то при увеличении данных в 10 раз, получаем увеличение времени в 100 раз. А при увеличении данных алгоритма умножения в 10 раз, мы получаем увеличение времени в 1000 раз соответственно. Примерно такой результат у на и выходит.

Ожидания по времени работы программы оправдались.

# 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. Бьерн Страуструп. Язык программирования С++ – Режим доступа: <http://8361.ru/6sem/books/Straustrup-Yazyk_programmirovaniya_c.pdf>
2. Т.А. Павловская Учебник по программированию на языках высокого уровня(С/С++) – Режим доступа: <http://cph.phys.spbu.ru/documents/First/books/7.pdf>
3. Совертков, П. И. Справочник по элементарной математике : учебное пособие / П. И. Совертков. — 2-е изд., стер. — Санкт-Петербург : Лань, 2022. — 404 с. — ISBN 978-5-8114-4132-7. — Текст : электронный // Лань : электронно-библиотечная система.
4. Сандаков, Е. Б. Векторная алгебра : учебно-методическое пособие / Е. Б. Сандаков, Ю. Н. Гордеев. — Москва : НИЯУ МИФИ, 2012. — 36 с. — ISBN 978-5-7262-1741-3. — Текст : электронный // Лань : электронно-библиотечная система.