Национальный исследовательский Нижегородский государственный университет имени Н. И. Лобачевского

Отчёт по лабораторной работе

Структуры хранения и операции над векторами и матрицами

Нижний Новгород

2023-2024 г.г.

Выполнил:  
студент ф-та ИИТММ, группа 3822Б1ФИ5  
Будыкин Рудольф-Евгений Александрович.

Проверил:  
Лебедев Илья Геннадьевич

Содержание:

1. [Введение](#Введение)
2. [Постановка задачи](#Постановка_задачи)
3. [Руководство пользователя](#Руководство_польз)
4. [Руководство программиста](#Руководство_прог)
5. [Описание структур данных](#Описание_структур)
6. [Описание алгоритмов](#Описание_алгоритмов)
7. [Тесты](#Тесты)
8. [Заключение](#Заключение)
9. [Литература](#Литература)
10. [Приложение](#Приложение)

**1. Введение**

Операции над векторами и матрицами являются весьма тривиальной задачей, но при большом количестве вычислений требуется облегчение процесса, ввиду невозможности держать в голове одновременно результаты всех действий даже над одним и тем же набором элементов.

**2. Постановка задачи**

Задачей является реализация программы (библиотеки) по работе с матрицами и векторами на основе классов. Обработка должна включать в себя операции с матрицами: умножение, сложение, вычитание, сравнение, аналогичные операции с векторами.

(Текущая версия программы (0.3) не имеет полного функционала, что указано в каждом из необходимых пунктов)

**3. Руководство пользователя (полная версия на релиз)**

Программа может выполнять определенные действия с векторами и матрицами, а именно:

• Создание и инициализация векторов и матриц разной размерности.

• Выполнение операций сложения, вычитания и умножения для векторов и матриц.

• Вывод результатов операций на экран.

**3.1 Руководство пользователя (версия 0.3)**

На данный момент реализованы следующие операции

* Сложение векторов
* Вычитание векторов
* Ввод/вывод результатов на экран отдельными функциями.

**4. Руководство программиста**

Программисту предоставляются библиотеки (файлы .h) включающие:

• TVector.h - Реализация класса для векторов. Операции над векторами.

• TMatrix.h - Класс для матриц, наследуемый от класса векторов. (в версии 0.3 не имеет поставленного задачей функционала)

• Тестовое приложение. После первого открытия кода в компиляторе необходимо выставить test\_MatrixLab в качестве запускаемого проекта. Приложение содержит ряд тестов демонстрирующих работу библиотек.

• Так же в проект включены source файлы TMatrix.cpp и TVector.cpp, необходимые для успешной сборки программы в cmake, функционала не несут ввиду полного включения оного в .h файлы.

**5. Описание структур данных**

1. Класс TVector и его элементы:

1. Поля класса:

* size – отвечает за размерность вектора.
* data – само содержание вектора (его числа по каждому из измерений).

1. Методы и конструкторы класса:

* Конструкторы по умолчанию, копирования, перемещения, перегрузки различных операторов для работы с векторами.
* Функции для непосредственной работы с данными вектора.

2. Класс TMatrix и его элементы

Для версии 0.3 реализован лишь как шаблон, не имеет смысла в представлении.

**6. Описание алгоритмов**

1. Оператор присваивания для TVector

• Алгоритм:

1. Проверка на самоприсваивание.
2. Проверка на непустое поле данных в присваиваемом векторе.
3. Обработка введенного вектора конвертером во избежание ошибок
4. Копирование элементов из исходного вектора в текущий.

2. Операторы сложения/вычитания для TVector

• Алгоритм:

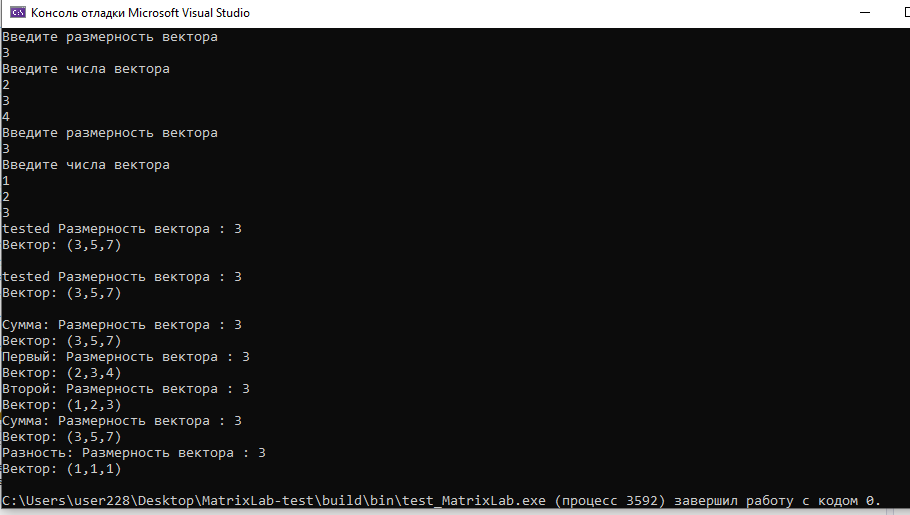
1. Проверка на совпадение размерности оперируемых векторов
2. Непосредственные операции над векторами, соответствующие оператору.

3. Функция Vvod() – ввод вектора с клавиатуры упрощающий код main файла для удобства программиста.

4. Функция AddVec отвечает за ошибки определения памяти (если вектор создан с одной размерностью, и по причине какой то ошибки у него оказалась другая, функция исправит это прогнав через себя все данные вектора), необходима для избавления от мусора в данных после функции ввода

5. Функции GetSize(), GetData() – дают доступ к данным извне класса

**7. Тесты**

Программа успешно проходит тесты по реализованным на данный момент функциям. 

**8. Заключение**

В лабораторной работе реализован не весь функционал но то что имеется в нынешней версии отрабатывает штатно. Программа может быть использована по прямому назначению для облегчения работы с векторами.

**9. Литература**

1. Лабораторный практикум. Составители Барышева. И.В, Мееров И.Б., Сысоев А.В., Шестакова Н.В. Под редакцией Гергеля В.П. Учебно-методическое пособие. – Нижний Новгород: Нижегородский госуниверситет, 2017. 105с.  
**10.Приложение**

#pragma once

#include <iostream>

template <class T>

class TVector

{

protected:

int size;

T\* data;

public:

TVector(const TVector<T>&);

TVector(TVector<T>&&);

TVector(int \_size = 3);

~TVector();

int GetSize();

T\* GetData();

void Resize(int);

TVector<T>operator+(const TVector<T>& p);

TVector<T>operator-(const TVector<T>& p);

TVector<T>&operator\*(const TVector<T>& p);

TVector<T>&operator\*(double p);

TVector<T>& operator=(const TVector<T>& p);

TVector<T>& AddVec(int \_size, T\* d);

friend std::ostream& operator<<(std::ostream& os, const TVector& p);

friend std::istream& operator>>(std::istream& is, TVector& p);

void Vvod();

};

template <class T>

TVector<T>::TVector(int \_size) //создание вектора по умолчанию

{

if (\_size <= 0)

throw "У вектора не может быть отрицательной размерности";

size = \_size;

data = new T[size];

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

{

data[i] = 0;

}

}

template <class T>

TVector<T>::TVector(const TVector<T>& p) //копирование вектора

{

size = p.size;

if (data != nullptr)

delete[] data;

data = new T[size];

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

data[i] = p.data[i];

}

template <class T>

TVector<T>::TVector(TVector<T>&& p) //перемещение вектора

{

size = p.size;

data = p.data;

p.size = 0;

p.data = nullptr;

}

template <class T>

TVector<T>::~TVector() //деструктор вектора

{

if (data != nullptr)

delete[]data;

data = nullptr;

size = 0;

}

template <class T>

int TVector<T>::GetSize() //получение размерности вектора

{

return size;

}

//Методы:

template <class T>

void TVector<T>::Resize(int newsize) // изменение размерности вектора

{

if (newsize <= 0)

throw "invalid size";

T\* newdata = new T[newsize];

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

{

newdata[i] = data[i];

delete[] data;

data = newdata;

}

}

template <class T>

T\* TVector<T>::GetData() // получение значений вектора

{

T\* res = new T;

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

{

res[i] = data[i];

}

return res;

}

template <class T>

TVector<T> TVector<T>::operator+ (const TVector <T>& p) //сложение векторов

{

if (size == p.size) //проверка совпадения размерности

{

TVector<T> res(size);

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

{

res.data[i] = data[i] + p.data[i];

}

std::cout << "tested " << res << std::endl;

return res;

}

if (size != p.size)

{

std::cerr << "Размерность векторов не совпадает";

throw -1;

}

}

template <class T>

TVector<T> TVector<T>::operator- (const TVector <T>& p) //сложение векторов

{

if (size == p.size) //проверка совпадения размерности

{

TVector<T> res(size);

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

{

res.data[i] = data[i] - p.data[i];

}

return res;

}

else

{

std::cerr << "Размерность векторов не совпадает";

}

}

/\*

template <class T>

TVector<T>& TVector<T>::operator\* (const TVector <T>& p)

{

}

\*/

template <class T>

TVector<T>& TVector<T>::operator\* (double chislo) //умножение вектора на число

{

TVector<T> res = \*this;

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

res.data[i] \*= chislo;

return res;

}

template <class T>

TVector<T>& TVector<T>::operator=(const TVector<T>& p) //присвоение вектора

{

if (this == &p)

return \*this;

if (data != nullptr)

delete[] data;

data = new T[size];

this->AddVec(size, data);

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

data[i] = p.data[i];

return \*this;

}

template <class T>

TVector<T>& TVector<T>::AddVec(int \_size, T\* d) //функция конвертирует входные данные в TVector

{

if (\_size != size)// по какой то вдруг причине размерности выделенной под вектор памяти и пришедшая не совпадают

{

delete[] data;

data = new T [\_size];

if (size < \_size) //Если пришедший более размерный чем выделенный

{

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

{

data[i] = d[i];

}

for (int i = size; i < \_size; i++)

{

data[i] = 0;

}

size = \_size;

return \*this;

}

if (size > \_size)// если пришедший менее размерный чем выделенный

{

for (int i = 0; i < \_size; i++)

{

data[i] = d[i];

}

size = \_size;

return \*this;

}

}

else

{

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

{

data[i] = d[i];

}

return \*this;

}

}

std::ostream& operator<<(std::ostream& os, const TVector<double> &p)

{

os << "Размерность вектора : " << p.size << std::endl;

os << "Вектор: (";

for (int i = 0; i < p.size; i++)

{

os << p.data[i];

if (i < p.size-1)

os << ",";

}

os << ")" << std::endl;

return os;

}

template <class T>

void TVector<T>::Vvod() // Функция ввода данных с клавиатуры

{

std::cout << "Введите размерность вектора" << std::endl;

std::cin >> size;

data = new T[size];

std::cout << "Введите числа вектора" << std::endl;

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

std::cin >> data[i];

this->AddVec(size, data); //отправка на конвертацию с проверкой (избавляет от ошибок, без нее в дате будет всякий бред)

}