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

Институт информационных технологий, математики и механики

Отчёт по лабораторной работе

Векторы

Выполнил:

студент ф-та ИТММ гр. 3821Б1ФИ3

Черепанов А.М

Проверил:

зав. лабораторией СТиВВ

Лебедев И.Г.

Нижний Новгород

2022 г

Содержание

[1. Введение 3](#_Toc39428061)

[2. Постановка задачи 4](#_Toc39428062)

[3. Руководство пользователя 5](#_Toc39428063)

[4. Руководство программиста 6](#_Toc39428064)

[4.1. Описание структуры программы 6](#_Toc39428065)

[4.2. Описание структур данных 6](#_Toc39428066)

[4.3. Описание алгоритмов 10](#_Toc39428067)

[5. Эксперименты 12](#_Toc39428068)

[6. Вывод 13](#_Toc39428069)

[7. Литература 14](#_Toc39428070)

[8. Приложения 15](#_Toc39428071)

1. **Введение**

Вектор и матрица в С++ выглядят как одномерный и двухмерный массив чисел. Для решения поставленной задачи необходимо использовать математические понятия с их прилегающими алгебраическими свойствами: сложение, умножение и другие математические действия.

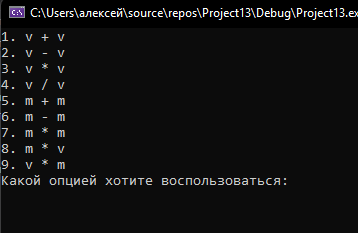
1. **Постановка задачи**

1) Написать шаблоны классов матрицы и вектора.

2) Продемонстрировать их работу на примере (main).

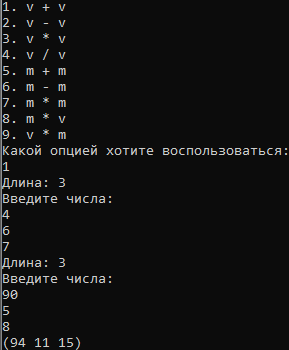
Должны быть:

* 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);
* метод деления матриц, метод, ищущий все вхождения заданного значения, метод, ищущий количество вхождений указанного значения;
* Сортировка вектора тремя способами (пузырек, вставка, быстрая сортировки).

1. **Руководство пользователя**
2. Выберите операцию

Пример 1 (рис. 1)

1. Следуйте указаниям в консоли – введите массив или вектор (зависит от сделанного выбора)



Пример 2 (рис. 2)

# Руководство программиста

### 4.1. Описание структуры программы

Программа состоит из одного решения.

В решении Vector определено 4 модуля main.cpp, Matrix.h, Vector.h.

* В модуле main.cpp определена стандартная функция int main(), где идет работа с остальными модулями.
* В модуле Matrix.h определен класс Matrix, а также объявлены все его методы и их определения.
* В модуле Vector.h определен класс Vector, а также объявлены все его методы и их определения.

### 4.2. Описание структур данных

В программе определены два следующих шаблонных класса:

1. Class TMatrix
2. Class TVector

Внутри класса Matrix определены следующие поля:

* int length – количество строк матрицы;
* int width– количество столбцов матрицы;

Внутри класса Matrix определен следующий набор public-методов

(плюс конструкторы и деструктор):

* TMatrix() – конструктор по умолчанию, не принимает никаких параметров, инициализирует все три поля 0, через списки инициализации;
* TMatrix(size\_t w, size\_t h)– конструктор инциализатор, принимает на вход два параметра типа size\_t;
* TMatrix(const TMatrix<T>& p) – конструктор копирования, принимает на вход объект типа TMatrix, создает объект с теми же характеристиками что и переданный, и матрицей того же содержимого;
* ~TMatrix() – деструктор, очищает выделенную методами и конструкторами память;
* int getWidth() – возвращает количество строк матрицы;
* int getHeight() – возвращает количество столбцов матрицы;
* TVector<T> operator[](const size\_t i) const– перегрузка оператора индексации, позволяет возвращать элемент матрицы по индексу;
* TMatrix<T> operator+(const TMatrix<T>& p) – перегрузка оператора суммы, позволяет возвращать матрицу, являющуюся суммой двух матриц;
* TMatrix<T> operator-(const TMatrix<T>& p) – перегрузка оператора разности, позволяет возвращать матрицу, являющуюся разностью двух матриц;
* TMatrix<T> operator=(const Matrix<T>& p) – перегрузка оператора присвоить, позволяет присваивать один объект типа Matrix другому;
* bool operator ==(const TMatrix<T>& p) – перегрузка оператора сравнения, позволяет сравнивать объекты типа Matrix;
* TMatrix<T> operator\*(const TMatrix<T>& p) – перегрузка оператора умножить, позволяет перемножать обьекты типа Matrix;
* friend ostream& operator<<(ostream& os, const TMatrix<T>& A)– перегрузка оператора вывода, позволяет выводить матрицы на экран;
* friend istream& operator>>(istream& in, TMatrix<T>& mat); – перегрузка оператора ввода, позволяет вводить матрицы.

Внутри класса Vector определены следующие поля:

* T\* data – шаблонный указатель.
* int len – количество элементов вектора.

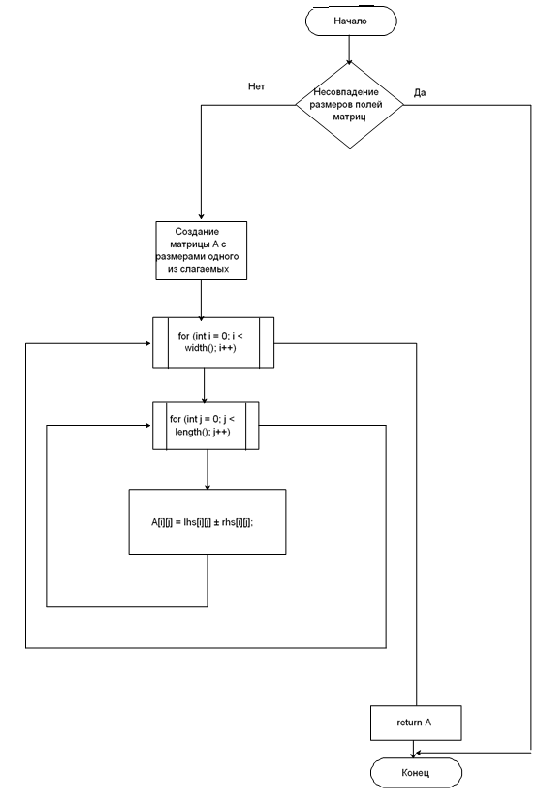
Внутри класса Vector определен следующий набор public-методов

(плюс конструкторы и деструктор):

* TVector() – конструктор по умолчанию, пустой.
* TVector(T\* data, const size\_t n)– конструктор инциализатор, принимает на вход параметр типа int и шаблонный указатель, создает динамический массив заданного размера, заполняет его теми же элементами какими заполнен переданный указатель;
* explicit TVector(const size\_t n) - конструктор инциализатор, принимает на вход параметр типа int, создает динамический массив заданного размера, заполняет его случайными числами.
* TVector(const Vector& p) – конструктор копирования, принимает на вход объект типа TVector, создает объект с теми же характеристиками что и переданный;
* ~TVector() – деструктор, очищает выделенную методами и конструкторами память;
* int size() – возвращает количество элементов вектора;
* T& operator [] (const size\_t i) const – перегрузка оператора индексации, позволяет возвращать элемент вектора по индексу;
* TVector<T> operator+(const TVector<T>& p) – перегрузка оператора суммы, позволяет возвращать вектор, являющийся суммой двух векторов;
* TVector<T> operator-(const TVector<T>& p) – перегрузка оператора разности, позволяет возвращать вектор, являющийся разностью двух векторов;
* TVector<T> operator = (const TVector<T>& p) – перегрузка оператора присвоить, позволяет присваивать один объект типа Vector другому;
* bool operator ==(const TVector<T>& p) – перегрузка оператора сравнения, позволяет сравнивать объекты типа Vector;
* TVector<T> operator \* (const TVector<T>& p) const – перегрузка оператора умножить, позволяет перемножать объекты типа Vector;
* friend ostream& operator<<(ostream& os, const TVector<T>& p)– перегрузка оператора вывода, позволяет выводить вектора на экран;
* friend istream& operator>>(istream& in, TVector<T>& p) – перегрузка оператора ввода, позволяет вводить вектора;

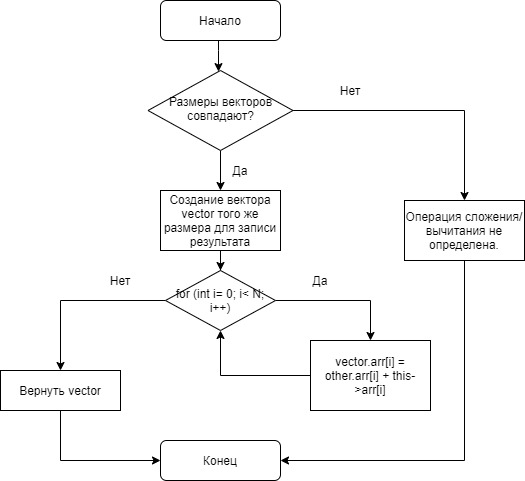
***4.3. Описание алгоритмов***

* Алгоритм суммирования или вычитания матриц:



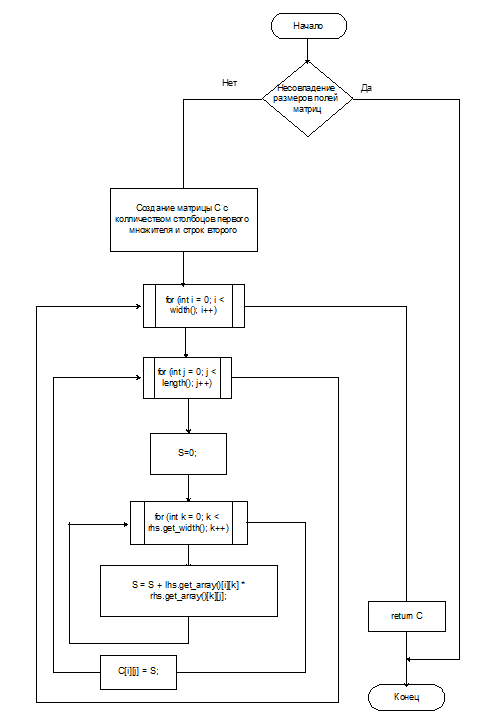
Блок-схема сложение/вычитание матриц (рис. 3)

* Алгоритм суммирования или вычитания векторов:



Блок-схема сложение/вычитание векторов (рис. 4)

* Алгоритм умножения матриц:



Блок-схема умножение матриц (рис. 6)

1. **Эксперименты**

Оценим время, которое занимают матричные, векторно-матричные операции, с помощью асимптотической сложности.

1. Сложение матриц:

|  |  |
| --- | --- |
| Количество элементов в матрице | Время выполнения |
| 10000 | 0 сек. |
| 1000000 | 0.09 сек |
| 100000000 | 0.53 сек. |

2. Умножение матриц.

|  |  |
| --- | --- |
| Количество элементов в матрице | Время выполнения |
| 100 | 0 сек. |
| 10000 | 0.007 сек. |
| 1000000 | 2.786 сек |

3. Умножение матриц и векторов.

|  |  |
| --- | --- |
| Количество элементов в матрице | Время выполнения |
| 100 | 0.00005 сек. |
| 1000000 | 0.543 сек. |
| 100000000 | 513.183 сек. |

1. **Вывод**

Удалось реализовать полноценную работу матриц и векторов, применить математические знания для реализации лабораторной работы.

1. **Литература**
2. Мейерс Скотт. Эффективный и современный С++;
3. Энтони Уильямс. Параллельное программирование на С++ в действии. Практика разработки многопоточных программ;

# Приложения

Vector. cpp

#include "Vector.h"

#include "Matrix.h"

main.cpp

#include<iostream>

#include "Vector.h"

#include "Matrix.h"

#include <time.h>

using namespace std;

int main()

{

TVector<int> v1, v2, v3;

TMatrix<int> m1, m2, m3;

setlocale(LC\_ALL, "RU");

int n;

cout << "1. v + v" << endl << "2. v - v" << endl << "3. v \* v" << endl << "4. v / v" << endl << "5. m + m" << endl << "6. m - m" << endl << "7. m \* m" << endl << "8. m \* v" << endl << "9. v \* m" << endl;

cout << "Какой опцией хотите воспользоваться:" << endl;

cin >> n;

switch (n)

{

case (1):

cin >> v1;

cin >> v2;

cout << v1 + v2;

return 0;

case(2):

cin >> v1;

cin >> v2;

cout << v1 - v2;

return 0;

case(3):

cin >> v1;

cin >> v2;

cout << v1 \* v2;

return 0;

case(4):

cin >> v1;

cin >> v2;

cout << v1 / v2;

return 0;

case(5):

cin >> m1;

cin >> m2;

cout << m1 + m2;

return 0;

case(6):

cin >> m1;

cin >> m2;

cout << m1 - m2;

return 0;

case(7):

cin >> m1;

cin >> m2;

cout << m1 \* m2;

return 0;

case(8):

cin >> m1;

cin >> v2;

cout << m1 \* v2;

return 0;

case(9):

cin >> v2;

cin >> m1;

cout << v2 \* m1;

return 0;

default:

break;

}

return 0;

}

Vector.h

#pragma once

#include <iostream>

#include <algorithm>

using namespace std;

template <typename T>

class TMatrix;

template <typename T>

class TVector

{

bool is\_own\_memory = true;

size\_t len = 0;

T\* data = nullptr;

protected:

T\* getData() const { return data; }

public:

TVector() {}

explicit TVector(const size\_t n);

TVector(const TVector<T>& p) { \*this = p; }

TVector(T\* data, const size\_t n) : len(n), data(data), is\_own\_memory(false) {}

~TVector() { if (is\_own\_memory) delete[] data; }

size\_t size() const { return len; }

T& operator [] (const size\_t i) const { return data[i]; }

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

bool operator != (const TVector<T>& p) const { return !(\*this == p); }

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

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

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

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

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

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

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

template <class T>

friend ostream& operator<<(ostream& os, const TVector<T>& p);

template <class T>

friend istream& operator>>(istream& in, TVector<T>& p);

};

template<class T>

inline TVector<T>::TVector(const size\_t n) : len(n)

{

if (len < 1) {

throw runtime\_error("Длина больше нуля!!!");

}

data = new T[len];

fill\_n(data, len, 0);

}

template<class T>

inline bool TVector<T>::operator==(const TVector<T>& p) const

{

if (size() != p.size())

return false;

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

{

if ((\*this)[i] != p[i])

return false;

}

return true;

}

template<class T>

inline TVector<T>& TVector<T>::operator=(const TVector<T>& p)

{

if (this == &p) {

return \*this;

}

if (data) {

delete[] data;

}

len = p.len;

data = new T[len];

copy\_n(p.data, len, data);

return \*this;

}

template<class T>

inline TVector<T> TVector<T>::operator+(const TVector<T>& p) const

{

if (size() != p.size())

throw runtime\_error("Размеры должны быть одинаковыми");

TVector<T> result(size());

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

result[i] = (\*this)[i] + p[i];

}

return result;

}

template<class T>

inline TVector<T> TVector<T>::operator-(const TVector<T>& p) const

{

if (size() != p.size())

throw runtime\_error("Размеры должны быть одинаковыми");

TVector<T> result(size());

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

result[i] = (\*this)[i] - p[i];

}

return result;

}

template<class T>

inline TVector<T> TVector<T>::operator+=(const TVector<T>& p)

{

\*this = \*this + p;

return \*this;

}

template<class T>

inline TVector<T> TVector<T>::operator/(const TVector<T>& p) const

{

if (size() != p.size())

throw runtime\_error("Размер умножаемых векторов должен быть одинаковым");

TVector<T> result(size());

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

result[i] = (\*this)[i] / p[i];

}

return result;

}

template<class T>

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

{

if (size() != p.size())

throw runtime\_error("Размер умножаемых векторов должен быть одинаковым");

TVector<T> result(size());

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

result[i] = (\*this)[i] \* p[i];

}

return result;

}

template<typename T>

inline TVector<T> TVector<T>::operator\*(const TMatrix<T>& matrix) const

{

if (matrix.getWidth() != size())

{

throw runtime\_error("Разные размеры");

}

TVector<T> result(size());

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

for (int j = 0; j < matrix.getWidth(); j++)

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

result[i] += (\*this)[i] \* matrix[k][i];

return result;

}

template<class T>

inline ostream& operator<<(ostream& os, const TVector<T>& p)

{

os << "(";

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

{

os << p.data[i];

if (i < p.len - 1)

os << " ";

}

os << ")";

return os;

}

template<class T>

inline istream& operator>>(istream& in, TVector<T>& p)

{

cout << "Длина: ";

in >> p.len;

if (p.len != 0) delete[] p.data;

p.data = new T[p.len];

cout << "Введите числа: " << endl;

for (int i = 0; i < p.len; i++) in >> p.data[i];

return in;

}

Multi.h

#pragma once

#include "Matrix.h"

#include "Vector.h"

template <class T>

TMatrix<T> operator \* (TVector<T>& vector, TMatrix<T>& matrix)

{

if ((matrix.GetL() == vector.GetW()))

{

T\*\* arr = new T \* [matrix.GetL()];

for (int i = 0; i < vector.GetW(); i++)

arr[i] = new T[vector.GetW()];

TMatrix<T> result(matrix.GetL(), vector.GetW(), arr);

for (int i = 0; i < result.GetW(); i++)

for (int j = 0; j < result.GetL(); j++)

{

result.SetI(i, j, matrix[0][j] \* vector[i]);

}

for (int i = 0; i < vector.GetW(); i++)

delete[] arr[i];

delete[] arr;

return result;

}

};

template <class T>

TVector<T> operator \*(TMatrix<T>& matrix, TVector<T>& vector)

{

if (matrix.GetW() == vector.GetW())

{

T\* arr = new T[matrix.GetL()];

TVector<T> result(matrix.GetL(), arr);

for (int i = 0; i < result.GetW(); i++)

{

result.SetI(i, 0);

for (int j = 0; j < vector.GetW(); j++)

{

result.SetI(i, result[i] + (matrix[i][j] \* vector[j]));

}

}

delete[] arr;

return result;

}

else

throw - 1;

};

Matrix.h

#pragma once

#include <iostream>

#include "Vector.h"

using namespace std;

template<typename T>

class TMatrix : public TVector<T>

{

size\_t width = 0;

size\_t height = 0;

public:

TMatrix() : TVector<T>() {}

TMatrix(size\_t w, size\_t h) : TVector<T>(w\* h), width(w), height(h) {}

TMatrix(const TMatrix<T>& p) { \*this = p; }

~TMatrix() {}

size\_t getWidth() const { return width; }

size\_t getHeight() const { return height; }

bool operator == (const TMatrix<T>& p) const;

bool operator != (const TMatrix<T>& p) const { return !(\*this == p); }

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

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

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

TMatrix<T> operator += (const TMatrix<T>& p);

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

TMatrix<T> operator / (const TMatrix<T>& p) const;

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

TVector<T> operator[](const size\_t i) const {

return TVector<T>(this->getData() + i \* width, width);

}

template <class T>

friend ostream& operator<<(ostream& os, const TMatrix<T>& A);

template<class T>

friend istream& operator>>(istream& in, TMatrix<T>& mat);

};

template<class T>

inline bool TMatrix<T>::operator==(const TMatrix<T>& p) const

{

if (p.width != width || p.height != height) {

return false;

}

return \*(static\_cast<TVector<T>\*>(this)) == \*(static\_cast<TVector<T>\*>(&p));

}

template<class T>

inline TMatrix<T>& TMatrix<T>::operator=(const TMatrix<T>& p)

{

if (this == &p)

return \*this;

width = p.width;

height = p.height;

\*(static\_cast<TVector<T>\*>(this)) = \*(static\_cast<const TVector<T>\*>(&p));

return \*this;

}

template<class T>

inline TMatrix<T> TMatrix<T>::operator+(const TMatrix<T>& p) const

{

if (getWidth() != p.getWidth() || getHeight() != p.getHeight())

throw runtime\_error("Shapes don't match");

TMatrix<T> result(getWidth(), getHeight());

\*(static\_cast<TVector<T>\*>(&result)) = \*(static\_cast<const TVector<T>\*>(this)) + \*(static\_cast<const TVector<T>\*>(&p));

return result;

}

template<class T>

inline TMatrix<T> TMatrix<T>::operator-(const TMatrix<T>& p) const

{

if (getWidth() != p.getWidth() || getHeight() != p.getHeight())

throw runtime\_error("Shapes don't match");

TMatrix<T> result(getWidth(), getHeight());

\*(static\_cast<TVector<T>\*>(&result)) = \*(static\_cast<const TVector<T>\*>(this)) - \*(static\_cast<const TVector<T>\*>(&p));

return result;

}

template<class T>

inline TMatrix<T> TMatrix<T>::operator+=(const TMatrix<T>& p)

{

\*this = \*this + p;

return \*this;

}

template<class T>

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

{

if (getWidth() == p.getHeight())

{

TMatrix<T> result(getHeight(), p.getWidth());

for (int i = 0; i < result.width; i++)

for (int j = 0; j < result.height; j++)

{

result[i][j] = 0;

for (int k = 0; k < height; k++)

result[i][j] = result[i][j] + ((\*this)[i][j] \* p[i][j]);

}

return result;

}

else

throw runtime\_error("shapes don't match");

}

template<typename T>

inline TMatrix<T> TMatrix<T>::operator/(const TMatrix<T>& p) const

{

if (getWidth() != p.getWidth() || getHeight() != p.getHeight())

throw runtime\_error("Shape of divided matrices must be the same");

TMatrix<T> result(getWidth(), getHeight());

\*(static\_cast<TVector<T>\*>(&result)) = \*(static\_cast<const TVector<T>\*>(this)) / \*(static\_cast<const TVector<T>\*>(&p));

return result;

}

template<typename T>

inline TVector<T> TMatrix<T>::operator\*(TVector<T>& vector) const

{

if (getWidth() != vector.size())

{

throw - 1;

}

TVector<T> result(vector.size());

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

for (int j = 0; j < getWidth(); j++)

for (int k = 0; k < vector.size(); k++)

result[i] += (\*this)[i][k] \* vector[k];

return result;

}

template<class T>

inline ostream& operator<<(ostream& B, const TMatrix<T>& A)

{

for (int i = 0; i < A.width; i++)

{

for (int j = 0; j < A.height; j++)

B << A[i][j] << " ";

cout << endl;

}

return B;

}

template<class T>

inline istream& operator>>(istream& in, TMatrix<T>& mat)

{

cout << "Введите ширину: ";

in >> mat.width;

cout << "Введите высоту: ";

in >> mat.height;

TMatrix<T> temp(mat.height, mat.width);

for (int i = 0; i < mat.width; i++)

for (int j = 0; j < mat.height; j++)

in >> temp[i][j];

mat = temp;

return in;

}