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

Канаков Р.А.

Проверил:

зав. лабораторией СТиВВ

Лебедев И.Г.

Нижний Новгород

2022 г

Оглавление

[Введение 3](#_Toc104492305)

[**1.** **Постановка задачи** 4](#_Toc104492306)

[**2.** **Руководство пользователя** 5](#_Toc104492307)

[**3.** **Руководство программиста** 7](#_Toc104492308)

[***3.1.*** ***Описание структуры данных программы*** 7](#_Toc104492309)

[***3.2.*** ***Описание структур данных*** 7](#_Toc104492310)

[**3.3.** **Описание алгоритмов** 10](#_Toc104492311)

[**4.** **Эксперимент** 11](#_Toc104492312)

[**5.** **Заключение** 12](#_Toc104492313)

[**6.** **Литература** 13](#_Toc104492314)

[**7.** **Приложения** 14](#_Toc104492315)

[**7.1.** **Приложение 1 (Main.cpp)** 14](#_Toc104492316)

[**7.2.** **Приложение 2 (Vector.h)** 15](#_Toc104492317)

[**7.3.** **Приложение 3 (Matrix.h)** 19](#_Toc104492318)

# Введение

В языке C++ вектор представляет собой одномерный динамический массив, а матрица – массив, элементами которого являются векторы. В данной работе необходимо описать операции сложения, умножения, деления и так далее, применимые для векторов и матриц.

1. **Постановка задачи**

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)

1. **Руководство пользователя**

После запуска программы, в консоли будет выведено сообщение:



Рис. 1. Ввод значений вектора

Программа создала вектор с размером 3, необходимо ввести 3 целых числа.

После ввода необходимо проделать ту же операцию для второго вектора.

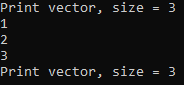


Рис.2. Ввод значений для второго вектора.

Затем программа выдаст значения проделанных операций.

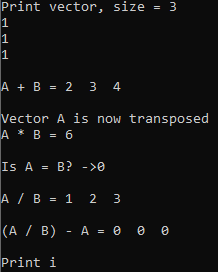


Рис.3. Вывод результатов вычислений.

Также, для умножения вектора A на B, применяется операция транспонирования (для вектора A). После этого, необходимо ввести число (в диапазоне от 0 до размера вектора).



Рис. 4. Вывод n-го элемента вектора.

Далее необходимо ввести 9 чисел, для заполнения матрицы 3\*3.

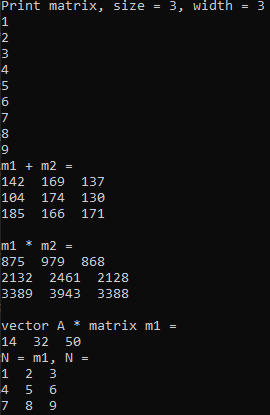


Рис. 5. Заполнение матрицы и вывод результатов вычислений.

Если все числа введены корректно, то после вывода результатов вычислений программа завершается.

1. **Руководство программиста**
   1. ***Описание структуры данных программы***

Программа состоит из двух проектов – vector и vectorlib. Решение Test содержит в себе 3 модуля: main.cpp, vector.h и matrix.h (также имеются файлы vector.cpp и matrix.cpp, но они не используются).

* В main.cpp находится функция int main(), где происходит работа с функциями, определенными в других модулях.
* В vector.h определен класс TVector, а также его поля и методы.
* В matrix.h определен класс TMatrix, который является наследником класса TVector, также здесь описаны поля и методы класса TMatrix.
  1. ***Описание структур данных***

Программа содержит два шаблонных класса: TVector и TMatrix.

Класс TVector содержит следующие поля:

* T\* data – указатель на массив типа “T”
* Int len – длина массива

Также этот класс содержит следующие публичные методы:

1. Конструктор по умолчанию

|  |
| --- |
| TVector(); |

1. Конструктор-инициализатор

|  |
| --- |
| TVector(int l, T v); |

1. Конструктор копирования

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

1. Деструктор

|  |
| --- |
| ~TVector(); |

1. Метод, получающий булевское значение, говорящее о том транспонирован вектор или нет

|  |
| --- |
| bool GetTransp(); |

1. Метод, изменяющий транспонированость вектора

|  |
| --- |
| void SetTransp(bool s); |

1. Метод, задающий длину вектора

|  |
| --- |
| void SetLen(int l); |

1. Метод, получающий длину вектора

|  |
| --- |
| int GetLen(); |

1. Метод, изменяющий длину вектора

|  |
| --- |
| void Resize(int l); |

1. Оператор +

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

1. Оператор –

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

1. Оператор \*

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

1. Оператор /

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

1. Оператор =

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

1. Оператор ==

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

1. Оператор индексации

|  |
| --- |
| T& operator [](int i); |

1. Оператор потока вывода

|  |
| --- |
| template <class T>  friend std::ostream& operator << (std::ostream& B, TVector<T>& p); |

1. Оператор потока ввода

|  |
| --- |
| template <class T>  friend std::istream& operator >> (std::istream& B, TVector<T>& p); |

Класс TMatrix содержит поле: int width – количество строк матрицы.

Также этот класс содержит следующие публичные методы:

1. Конструктор по умолчанию

|  |
| --- |
| TMatrix(); |

1. Конструктор инициализатор 1

|  |
| --- |
| TMatrix(int l); |

1. Конструктор инициализатор 2

|  |
| --- |
| TMatrix(int w, int l); |

1. Конструктор копирования

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

1. Деструктор

|  |
| --- |
| ~TMatrix(); |

1. Оператор +

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

1. Оператор –

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

1. Оператор \*(матрица-матрица)

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

1. Оператор \*(матрица-вектор)

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

1. Оператор =

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

1. Оператор ==

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

1. Оператор индексации

|  |
| --- |
| TVector<T>& operator [] (const int i); |

1. Метод, получающий количество строк матрицы

|  |
| --- |
| int GetWidth(); |

1. Метод, изменяющий количество строк матрицы

|  |
| --- |
| void SetWidth(int w); |

1. Метод, изменяющий размер матрицы

|  |
| --- |
| void Resize(int w, int l); |

1. Оператор потока вывода

|  |
| --- |
| template <class T>  friend std::ostream& operator << (std::ostream& B, TMatrix<T>& A); |

1. Оператор потока ввода

|  |
| --- |
| template <class T>  friend std::istream& operator >> (std::istream& B, TMatrix<T>& A); |

* 1. **Описание алгоритмов**

1. **Эксперимент**
2. **Заключение**

В данной работе удалось реализовать работу вектор и матриц, а также операций над ними.

Продемонстрирована работоспособность программы.

Программа имеет:

* 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)

1. **Литература**

* Лекции Лебедева Ильи Геннадьевича

<https://cloud.unn.ru/s/tYGZ4NDkCgx6mRq>

<https://cloud.unn.ru/s/fj8ja3ZgfZY3HWa>

* Язык программирования C++

<https://ru.wikipedia.org/wiki/C%2B%2B>

1. **Приложения**
   1. **Приложение 1 (Main.cpp)**

#include <iostream>

#include "Vector.h"

#include "Matrix.h"

int main()

{

TVector<int> A(3, 0);

TVector<int> B(3, 0);

TVector<int> C(3, 0);

int a = 0;

std::cin >> A >> B;

std::cout << std::endl;

C = A + B;

std::cout << "A + B = " << C << std::endl;

A.SetTransp(true);

std::cout << "Vector A is now transposed" << std::endl;

a = A \* B;

std::cout << "A \* B = " << a << std::endl << std::endl;

std::cout << "Is A = B? ->" << (A == B) << std::endl << std::endl;

C = A / B;

std::cout << "A / B = " << C << std::endl;

C = C - A;

std::cout << "(A / B) - A = " << C << std::endl;

int i = 0;

std::cout << "Print i ";

std::cin >> i;

std::cout << "A[i] = " << A[i] << std::endl;

TMatrix<int> m1(3, 3);

TMatrix<int> m2(3, 3);

TMatrix<int> m3(3, 3);

for (int j = 0; j < m2.GetLen(); j++)

for (int k = 0; k < m2.GetWidth(); k++)

m2[j][k] = rand() % 100 + 100;

std::cin >> m1;

m3 = m1 + m2;

std::cout << "m1 + m2 = " << std::endl << m3 << std::endl;

m3 = m1 \* m2;

std::cout << "m1 \* m2 = " << std::endl << m3 << std::endl;

TVector<int> M(3, 0);

M = m1 \* A;

std::cout << "vector A \* matrix m1 = " << std::endl;

std::cout << M;

TMatrix<int> N(3, 3);

N = m1;

std::cout << "N = m1, N = " << std::endl << N;

return 0;

}

* 1. **Приложение 2 (Vector.h)**

#pragma once

#include <iostream>

template <class T>

class TVector

{

public:

TVector();

TVector(int l, T v);

TVector(const TVector<T>& p);

~TVector();

bool GetTransp();

void SetTransp(bool s);

void SetLen(int l);

int GetLen();

void Resize(int l);

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

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

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

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

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

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

T& operator [](int i);

template <class T>

friend std::ostream& operator << (std::ostream& B, TVector<T>& p);

template <class T>

friend std::istream& operator >> (std::istream& B, TVector<T>& p);

protected:

T\* data;

int len;

bool transp;

};

template<class T>

inline TVector<T>::TVector()

{

this->len = 0;

this->data = nullptr;

this->transp = false;

}

template<class T>

inline TVector<T>::TVector(int l, T v)

{

if (l < 1)

throw "len < 1";

this->transp = false;

this->data = new T[l];

this->len = l;

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

this->data[i] = v;

}

template<class T>

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

{

if (p.data == nullptr)

{

this->data = nullptr;

this->len = 0;

this->transp = false;

}

else

{

this->len = p.len;

this->transp = false;

this->data = new T[len];

for (int i = 0; i < this->len; i++)

this->data[i] = p.data[i];

}

}

template<class T>

inline TVector<T>::~TVector()

{

if (this->data != nullptr)

{

delete[] this->data;

this->data = nullptr;

}

this->len = 0;

}

template<class T>

inline bool TVector<T>::GetTransp()

{

return transp;

}

template<class T>

inline void TVector<T>::SetTransp(bool s)

{

this->transp = s;

}

template<class T>

inline void TVector<T>::SetLen(int l)

{

this->Resize(l);

this->len = l;

}

template<class T>

inline int TVector<T>::GetLen()

{

return len;

}

template<class T>

inline void TVector<T>::Resize(int l)

{

if (l < 1)

throw "len < 1";

T\* mas = new T[l];

if (this->data != nullptr)

{

int a = 0;

if (this->len > l)

a = l;

else a = this->len;

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

mas[i] = this->data[i];

delete[] this->data;

}

this->data = mas;

this->len = l;

}

template<class T>

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

{

if (this->len == 0)

throw "len = 0";

if (this->len != p.len)

throw "len isn't the same";

TVector<T> res(this->len, 0);

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

res[i] = this->data[i] + p.data[i];

return res;

}

template<class T>

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

{

if (this->len == 0)

throw "len = 0";

if (this->len != p.len)

throw "len isn't the same";

TVector<T> res(this->len, 0);

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

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

return res;

}

template<class T>

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

{

if (this->len == 0)

throw "len = 0";

if (this->len != p.len)

throw "len isn't the same";

int a = 0;

if ((p.transp == true && this->transp == false) || (this->transp == true && p.transp == false))

{

for (int i = 0; i < this->len; i++)

{

a += (\*this)[i] \* p.data[i];

}

}

else

throw "Error: one vector must be transposed";

return a;

}

template<class T>

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

{

if (this->len == 0)

throw "len = 0";

if (this->len != p.len)

throw "len isn't the same";

TVector<T> A(this->len, 0);

for (int i = 0; i < this->len; i++)

{

if (p.data[i] == 0)

A[i] = 0;

else

A[i] = this->data[i] / p.data[i];

}

return A;

}

template<class T>

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

{

if (this == &p)

return \*this;

if (this->data != nullptr)

delete[] this->data;

if (p.data == nullptr)

{

this->len = 0;

this->data = nullptr;

}

else

{

this->len = p.len;

this->data = new T[len];

for (int i = 0; i < this->len; i++)

this->data[i] = p.data[i];

}

return \*this;

}

template<class T>

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

{

if (this->len != p.len)

return false;

for (int i = 0; i < this->len; i++)

{

if (this->data[i] != p.data[i])

return false;

}

return true;

}

template<class T>

inline T& TVector<T>::operator[](int i)

{

if (data == nullptr)

throw "data = 0";

if (i < 0 || i >= len)

throw "i is out of range";

return data[i];

}

template<class T>

inline std::ostream& operator<<(std::ostream& B, TVector<T>& p)

{

if (p.GetTransp() == false)

{

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

B << p[i] << " ";

}

else

{

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

B << p[i] << std::endl;

}

B << std::endl;

return B;

}

template<class T>

inline std::istream& operator>>(std::istream& B, TVector<T>& p)

{

std::cout << "Print vector" << ", size = " << p.len << std::endl;

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

B >> p[i];

return B;

}

* 1. **Приложение 3 (Matrix.h)**

#pragma once

#include "Vector.h"

template <class T>

class TMatrix : public TVector<TVector<T>>

{

public:

TMatrix();

TMatrix(int l);

TMatrix(int w, int l);

TMatrix(const TMatrix<T>& p);

~TMatrix();

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

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

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

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

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

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

TVector<T>& operator [] (const int i);

int GetWidth();

void SetWidth(int w);

void Resize(int w, int l);

template <class T>

friend std::ostream& operator << (std::ostream& B, TMatrix<T>& A);

template <class T>

friend std::istream& operator >> (std::istream& B, TMatrix<T>& A);

protected:

int width;

};

template<class T>

inline TMatrix<T>::TMatrix()

{

width = 0;

this->data = nullptr;

}

template<class T>

inline TMatrix<T>::TMatrix(int l) : TMatrix()

{

if (l < 0) throw "Error";

this->len = l;

width = l;

this->data = new TVector<T>[width];

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

this->data[i].Resize(len);

}

template<class T>

inline TMatrix<T>::TMatrix(int w, int l) : TMatrix()

{

if (l < 0 || w < 0)

throw "len < 0 or width < 0";

this->len = l;

width = w;

this->data = new TVector<T>[width];

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

this->data[i].Resize(len);

}

template<class T>

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

{

this->len = p.len;

width = p.width;

if (p.data == nullptr)

this->data = nullptr;

else

{

this->data = new TVector<T>[width];

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

this->data[i] = p.data[i];

}

}

template<class T>

inline TMatrix<T>::~TMatrix()

{

if (this->data != nullptr)

{

delete[] this->data;

this->data = nullptr;

}

width = 0;

}

template<class T>

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

{

if (this->len != p.len || width != p.width)

throw "len or width aren't the same";

TMatrix<T> res(\*this);

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

res.data[i] = this->data[i] + p.data[i];

return res;

}

template<class T>

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

{

if (this->len != p.len || width != p.width)

throw "len or width aren't the same";

TMatrix<T> res(width, this->len);

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

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

return res;

}

template<class T>

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

{

if (width != p.len)

throw "1st matrix width != 2nd matrix len";

TMatrix<T> res(width, p.len);

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

{

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

{

res[i][j] = 0;

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

{

res[i][j] += this->data[i][k] \* p.data[k][j];

}

}

}

return res;

}

template<class T>

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

{

if (width != p.GetLen())

throw "matrix width != vector len";

if (p.GetTransp() == false)

throw "vector isn't transposed";

TVector<T> res(width, 0);

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

{

for (int j = 0; j < this->len; j++)

{

res[i] += (this->data)[i][j] \* p[j];

}

}

res.SetTransp(true);

return res;

}

template<class T>

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

{

if (this == &p)

return \*this;

this->len = p.len;

width = p.width;

if (this->data != nullptr)

delete[] data;

if (p.data == nullptr)

data = nullptr;

else

data = new TVector<T>[p.width];

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

this->data[i] = p.data[i];

return \*this;

}

template<class T>

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

{

if (this->len != p.len || width != p.width)

return false;

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

{

for (int j = 0; j < this->len; j++)

if (this->data[i][j] != this->p.data[i][j]) return false;

}

return true;

}

template<class T>

inline TVector<T>& TMatrix<T>::operator[](const int i)

{

if (i < 0 || i >= width)

throw "i is out of range";

return this->data[i];

}

template<class T>

inline int TMatrix<T>::GetWidth()

{

return width;

}

template<class T>

inline void TMatrix<T>::SetWidth(int w)

{

this->Resize(this->len, w);

}

template<class T>

inline void TMatrix<T>::Resize(int w, int l)

{

TVector<T>\* res = new TVector<T>[w];

if (this->data != nullptr)

{

int w1, l1;

if (width > w)

w1 = w;

else

w1 = width;

if (this->len > l)

l1 = l;

else

l1 = this->len;

for (int i = 0; i < w1; i++)

{

res[i].Resize(l);

for (int j = 0; j < l1; j++)

res[i][j] = this->data[i][j];

}

delete[] data;

}

this->data = res;

this->len = l;

width = w;

}

template<class T>

inline std::ostream& operator<<(std::ostream& B, TMatrix<T>& p)

{

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

{

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

{

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

}

B << std::endl;

}

return B;

}

template<class T>

inline std::istream& operator>>(std::istream& B, TMatrix<T>& p)

{

std::cout << "Print matrix, size = " << p.GetLen() << ", width = " << p.GetWidth() << std::endl;

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

{

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

{

B >> p.data[i][j];

}

}

return B;

}