Федеральное агентство по образованию Российской Федерации

Государственное образовательное учреждение

высшего профессионального образования

Нижегородский государственный университет им. Н.И. Лобачевского

Институт информационных технологий математики и механики

Направление подготовки: «Фундаментальная информатика и информационные технологии»

Отчёт по лабораторной работе

**Реализация классов для работы с матрицами и векторами.**

Выполнил:

студент ИИТММ гр. 381906-2

Кулемин П. А.

Проверил:

ассистент каф. математического обеспечения

и суперкомпьютерных технологий

Лебедев И.Г.

Нижний Новгород

2019 г.

Содержание

I. [Введение 3](#_Toc24457830)

II. [Постановка задачи](#_Toc24457831) 4

III. [Руководство пользователя](#_Toc24457832) 5

IV. [Руководство программиста](#_Toc24457833) 6

1. [Описание структуры программы](#_Toc24457834) 6

2. [Описание структур данных](#_Toc24457835) 7

3. [Описание алгоритмов](#_Toc24457836) 7

V. [Эксперименты](#_Toc24457837) 10

VI. [Заключение](#_Toc24457838) 11

VII. [Литература](#_Toc24457839) 12

VIII. [Приложение 1](#_Toc24457840)3

# 1. Введение

Матрицы и векторы, это важная часть векторной алгебры, и поэтому информатизация и автоматизация в этой области математики крайне важна. Именно для этого мной были созданы классы для работы с векторами и матрицами, а также произведена теоретическая и практическая оценка времени работы различных арифметических операций над матрицами, а также времени различных сортировок, которые можно произвести над векторами. Также было произведено сравнение этих оценок.

# 2. Постановка задачи

Реализовать классы для работы с векторами и матрицами использовать шаблоны. Продемонстрировать их работу на примере (написать в main пример).

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).
* в классе вектор должна быть возможность отсортировать его тремя способами.

Сравнить время работы, и сделать выводы.

# 3. Руководство пользователя

Шаги, совершаемые пользователем:

1. Создать объект типа Matrix или Vector, в зависимости от необходимости.
2. Расширить матрицу или вектор на нужное количество элементов, используя конструкторы либо, методы доступа к защищенным полям.
3. Произвести необходимые математические операции.

# 4. Руководство программиста

## 4.1. Описание структуры программы

Программа состоит из одного решения.

В решении Vector определен 1 модуля: main.c. В модуле main.c определена стандартная функция int main(), внутри которой содержится набор действий с объектами типа Matrix и Vector.

## Описание структур данных

В программе определены два следующих шаблонных класса:

Class Matrix

Class Vector

Внутри класса Matrix определены следующие поля:

* Type\*\* matrix – шаблонный двойной указатель.
* int width – целочисленный размер ширины матрицы
* int length – целочисленный размер длины матрицы

Внутри класса Matrix определен следующий набор public-методов

(плюс конструкторы и деструктор):

* Matrix() – конструктор по умолчанию, не принимает никаких параметров, инициализирует все три поля 0, через списки инициализации.
* Matrix(int, int) – конструктор инциализатор, принимает на вход два параметра типа int, создает динамический массив заданных размеров, заполняет его 0, поля width и length, становятся значениями переданными в конструктор соответственно.
* Matrix(const Matrix& lhs) – конструктор копирования, принимает на вход объект типа Matrix, создает объект с теми же характеристиками что и переданный, и матрицей того же содержимого.
* Virtual ~Matrix() – виртуальный деструктор, очищает выделенную методами и конструкторами память.
* Int get\_width() – метод возвращающий ширину матрицы
* Int get\_length() – метод возвращающий длину матрицы
* Type\*\* get\_array() – метод возвращающий массив содержащий в себе все элементы матрицы.
* Void set\_width\_length(int,int) – метод принимающий новые размеры матрицы, приводящий матрицу к этим размерам. В случае если новая матрица больше старой, новые элементы равняются нулю. В противном случае, все элементы на которые хватило места, останутся в матрице. В случае если матрица имеет любую размерность равную 0, метод ведет себя аналогично конструктору инциализатору
* Type\*& operator [ ](int) - перегрузка оператора индексации, позволяет возвращать элемент матрицы по индексу.
* Friend Matrix operator +(Matrix & lhs, Matrix & rhs) – перегрузка оператора суммы, позволяет возвращать матрицу, являющуюся суммой двух матриц.
* Friend Matrix operator –(Matrix& lhs, Matrix& rhs) – перегрузка оператора –, аналогична в своей сути перегрузке оператора +
* Matrix& operator =(const Matrix & lhs) - перегрузка оператора присвоить, позволяет присваивать один объект типа Matrix другому
* friend bool operator ==(Matrix& lhs, Matrix& rhs) – перегрузка оператора сравнения, позволяет сравнивать объекты типа Matrix
* friend Matrix operator \*(const Matrix& lhs, const Matrix& rhs) - перегрузка оператора умножить, позволяет перемножать обьекты типа Matrix
* friend ostream& operator <<(ostream& out, const Matrix& rhs) – перегрузка оператора вывода, позволяет выводить матрицы на экран
* friend istream& operator >>(istream& in, Matrix& rhs) - перегрузка оператора ввода, позволяет вводить матрицы уже заданного размера, либо задавать размер и вводить самостоятельно.

Единственным «Private» методом, является метод

* void MemorySize(int, int) –выделяет память до размеров матрицы переданных в функцию

Шаблонный класс Vector, является public, наследником класса Matrix, поэтому никаких полей в этом классе не определено. Также почти 70% методов, он наследует от класса-родителя, поэтому внутри него, определены только специфичные для него методы:

* T1\* get\_vector() - метод возвращающий массив содержащий в себе все элементы объекта Vector.
* T1 operator [](const int) - перегрузка оператора индексации, позволяет возвращать элемент вектора по индексу.
* void set\_length(int) - метод принимающий новые размеры вектора, приводящий вектор к этим размерам. В случае если новый вектор больше старого, новые элементы равняются нулю. В противном случае, все элементы на которые хватило места, останутся в векторе. В случае если вектор имеет размерность равную 0, метод ведет себя аналогично конструктору инциализатору
* friend Vector operator /(Vector& lhs, Vector& rhs) – перегрузка оператора / возвращает вектор, координаты которого являются частным от деления двух соответствующих координат операндов. В случае, если делитель равен нулю, значение частного считается равным делимому.
* friend float operator \*(Vector& lhs, Vector & rhs) – перегрузка оператора умножить, возвращает скалярное произведение операндов.
* friend istream& operator >>(istream& in, Vector& rhs) - перегрузка оператора ввода, позволяет вводить матрицы уже заданного размера, либо задавать размер и вводить самостоятельно.
* clock\_t BubbleSort() - сортировка методом «Пузырек». Возвращает разницу между тактами процессора при входе в сортировку и при выходе из нее.
* clock\_t InsertionSort() - сортировка методом «Вставка». Возвращает разницу между тактами процессора при входе в сортировку и при выходе из нее.

## Описание алгоритмов

* Алгоритм суммирования или вычитания матриц:

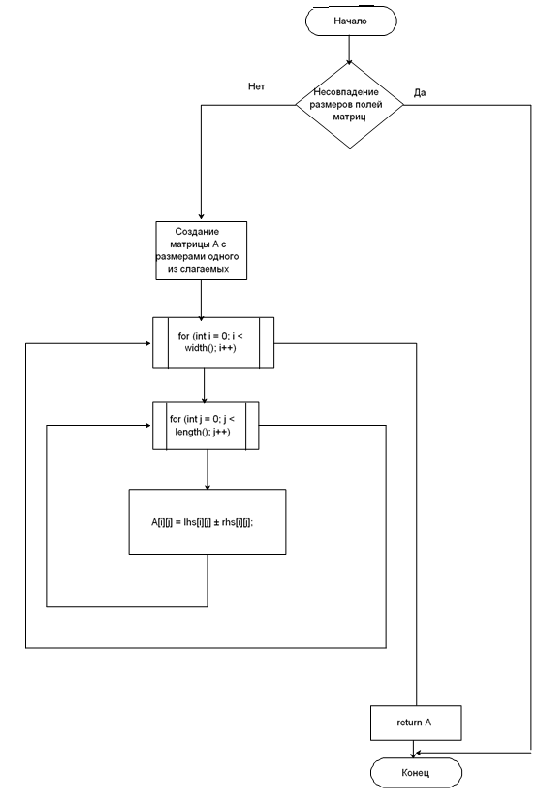


Рисунок 1: Блок-схема алгоритма сложения/вычитания матриц

# Алгоритм умножения матрицы на матрицу:

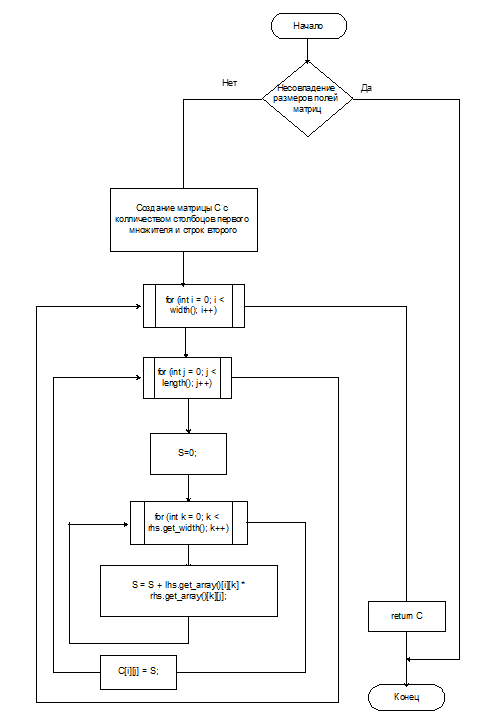


Рисунок 2: Блок-схема алгоритма умножения матриц.

Поскольку класс Vector это публичный наследник класса Matrix, то для него нет смысла описывать каким-либо образом арифметическую операцию сложения, так как эта операция в числе других, наследуется от класса Matrix.

* Алгоритм деления двух векторов:

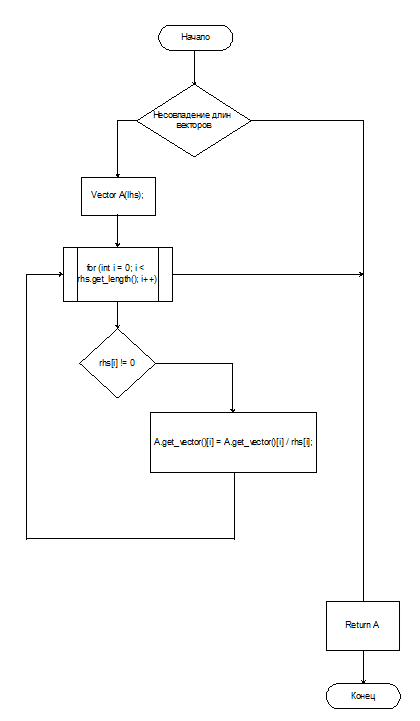


Рисунок 3: Блок-схема умножения одной матрицы на другою.

* Алгоритм умножения двух векторов:

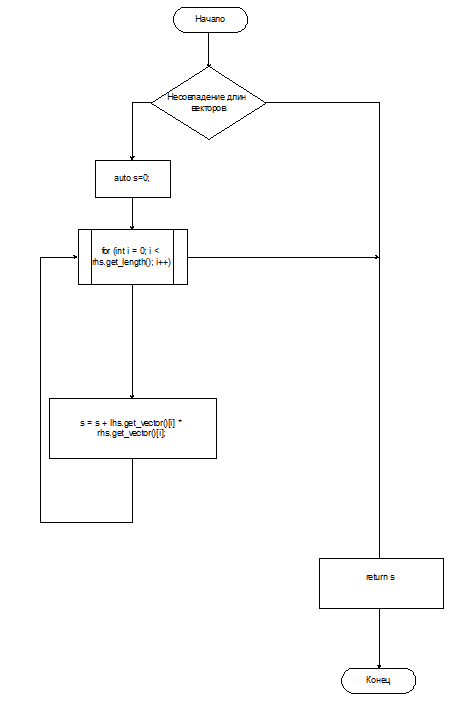


Рисунок 4: Блок-схема умножения двух векторов.

# Эксперименты

Оценим время, которое занимают матричные и векторно-матричные операции. Для этого мы произведем замеры времени умножения и сложения квадратных матриц (чтобы была возможность построить график) а также время умножения таких матриц на вектор.

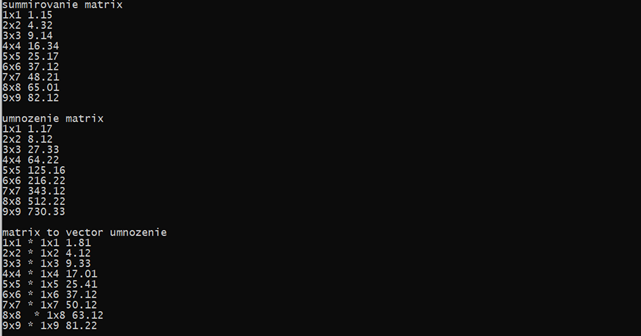


Рисунок 5: замеры времени, полученные при выполнении различных арифм. операций.

На основе полученных данных, построим графики зависимости времени от размеров матриц или векторов. А затем сравним их с графиками, теоретической сложности, которые легко построить, если проанализировать код, и определить его теоретическую сложность.

График 1: Соотношение теоретического и практического времени умножения.

График 2: Соотношение теоретического и практического времени.

Помимо этого, произведем исследование времени сортировки векторов. Для этого:



Рисунок 6. Пример работы программы с входными массивами на 15000

|  |  |  |
| --- | --- | --- |
| Вид сортировки | Время теоретическое (сек.) | Время практическое (сек.) |
| BubbleSort | 28,270 | 18,3 |
| InsertionSort | 8,606 | 8,234 |
| QuickSort | 0,024 | 0,025 |

Таблица 1. Результаты сортировок по времени для массива на 15000 элементов

Как мы можем видеть, при одном и том же входном массиве время сортировок из второй Лабораторной работы, близко ко времени которое получили мы. Таким образом мы подтвердили наше предположение.

# 6. Заключение

По итогу проведения данной работы, нам удалось реализовать удобные объекты для работы с матрицами и векторами. Также было установлено, что наиболее быстрой является сортировка QuickSort, следующей InsertionSort, а самой медленной BubleSort.

# 7. Литература

1. Павловская Т.А. C/C++, Программирование на языке высокого уровня, 2003.
2. <https://ru.wikipedia.org>
3. <https://ru.stackoverflow.com>
4. <https://learnc.info/c/memory_allocation.html>

# 8. Приложение

#include <cstdlib>

#include <iostream>

#include <malloc.h>

#include <time.h>

using namespace std;

template <class Type>

class Matrix

{

public:

Matrix() :width(0), length(0), matrix(0){}

Matrix(int \_a, int \_b) : width(\_a), length(\_b){

matrix = new Type\*[\_a];

MemorySize(\_a, \_b);

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

{

for (int j = 0; j < \_b; j++)

{

matrix[i][j] = 0;

}

}

}

Matrix(const Matrix& lhs) :width(lhs.width), length(lhs.length)

{

matrix = new Type\*[lhs.width];

MemorySize(lhs.width, lhs.length);

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

{

for (int j = 0; j < lhs.length; j++)

{

matrix[i][j] = lhs.matrix[i][j];

}

}

}

virtual ~Matrix()

{

for (int i = 0; i < width; i++) delete matrix[i];

delete matrix;

}

int get\_width()const

{

return width;

}

int get\_length()const

{

return length;

}

Type\*\* get\_array()

{

return matrix;

}

void set\_width\_length(int \_width, int \_length)

{

Type \*\* vr = new Type \*[width];

for (int i = 0; i < width; i++) vr[i] = new Type[length];

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

{

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

{

vr[i][j] = matrix[i][j];

}

}

if (width == 0)matrix = new Type\*[\_width];

MemorySize(\_width, \_length);

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

{

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

{

matrix[i][j] = vr[i][j];

}

}

if ((\_width > width) && (\_length>length))

{

for (int i = width; i < \_width; i++)

{

for (int j = length; j < \_length; j++)

{

matrix[i][j] = 0;

}

}

}

else{

if (\_width > width)

{

for (int i = width; i < \_width; i++)

{

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

{

matrix[i][j] = 0;

}

}

}

if (\_length > length)

{

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

{

for (int j = length; j < \_length; j++)

{

matrix[i][j] = 0;

}

}

}

}

width = \_width;

length = \_length;

}

Type\*& operator [](const int i)

{

return matrix[i];

}

friend Matrix operator +(Matrix& lhs, Matrix& rhs)

{

Matrix A(lhs.get\_width(), rhs.get\_length());

for (int i = 0; i < rhs.get\_width(); i++)

{

for (int j = 0; j < rhs.get\_length(); j++)

{

A[i][j] = lhs[i][j] + rhs[i][j];

}

}

return A;

}

friend Matrix operator -(Matrix& lhs, Matrix& rhs)

{

Matrix A(lhs);

for (int i = 0; i < rhs.get\_width(); i++)

{

for (int j = 0; j < rhs.get\_length(); j++)

{

A[i][j] = A[i][j] - rhs[i][j];

}

}

return A;

}

Matrix& operator =(Matrix& lhs)

{

width = lhs.get\_width();

length = lhs.get\_length();

matrix = new Type\* [width];

MemorySize(width, length);

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

{

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

{

matrix[i][j] = lhs[i][j];

}

}

return \*this;

}

friend bool operator ==(Matrix& lhs, Matrix& rhs)

{

if ((lhs.get\_width() != rhs.get\_width()) || (lhs.get\_length() != rhs.get\_length()))

{

return false;

}

else{

for (int i = 0; i < lhs.get\_width(); i++)

{

for (int j = 0; j < lhs.get\_length(); j++)

{

if (lhs[i][j] != rhs[i][j]) return false;

}

}

}

return true;

}

friend Matrix operator \*(const Matrix& lhs, const Matrix& rhs)

{

Matrix C(lhs.get\_width(), rhs.get\_length());

Type S;

for (int i = 0; i < lhs.get\_width(); i++)

{

for (int j = 0; j < rhs.get\_length(); j++)

{

S = 0;

for (int k = 0; k < rhs.get\_width(); k++)

{

S = S + lhs.get\_array()[i][k] \* rhs.get\_array()[k][j];

}

C[i][j] = S;

}

}

return C;

}

friend ostream& operator <<(ostream& out, Matrix& rhs)

{

for (int i = 0; i < rhs.get\_width(); i++)

{

for (int j = 0; j < rhs.get\_length(); j++)

{

out << rhs.get\_array()[i][j] << " ";

}

out << endl;

}

return out;

}

friend istream& operator >>(istream& in, Matrix& rhs)

{

if ((rhs.get\_length() == 0) || (rhs.get\_width() == 0))

{

cout << "Vvedite width&Length" << endl;

int a, b;

in >> a >> b;

rhs.set\_width\_length(a, b);

}

for (int i = 0; i < rhs.get\_width(); i++)

{

for (int j = 0; j < rhs.get\_length(); j++)

{

in >> rhs[i][j];

}

}

return in;

}

private:

void MemorySize(int \_width, int \_length)

{

for (int i = 0; i < \_width; i++) matrix[i] = new Type[\_length];

}

Type\*\* matrix;

int width;

int length;

};

template <class T1>

class Vector :public Matrix<T1>

{

public:

Vector() :Matrix(){}

Vector(int \_size) :Matrix(1, \_size){}

Vector(const Vector& t) :Matrix(t){}

~Vector(){ }

T1\* get\_vector()

{

return get\_array()[0];

}

using Matrix::operator =;

T1& operator [](const int i)

{

return get\_vector()[i];

}

void set\_length(int \_resize)

{

set\_width\_length(1, \_resize);

}

friend Vector operator /(Vector& lhs, Vector& rhs)

{

Vector A(lhs);

for (int i = 0; i < A.get\_length(); i++){

if (rhs[i] != 0){

A.get\_vector()[i] = A.get\_vector()[i] / rhs[i];

}

}

return A;

}

friend float operator \*(Vector& lhs, Vector & rhs)

{

float s = 0;

for (int i = 0; i < rhs.get\_length(); i++)s = s + lhs.get\_vector()[i] \* rhs.get\_vector()[i];

return s;

}

friend istream& operator >>(istream& in, Vector& rhs)

{

if (rhs.get\_length() == 0)

{

int \_size;

in >> \_size;

rhs.set\_length(\_size);

}

for (int i = 0; i < rhs.get\_length(); i++)

{

in >> rhs.get\_vector()[i];

}

return in;

}

friend clock\_t BubbleSort(Vector &A)

{

clock\_t t1, t2;

t1 = clock();

T1 tmp = 0;

int i, j = 0;

t1 = clock();

for (int i = 0; i <A.get\_length(); i++)

{

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

{

if (A.get\_vector()[j] > A.get\_vector()[j + 1])

{

tmp = A.get\_vector()[j];

A.get\_vector()[j] = A.get\_vector()[j + 1];

A.get\_vector()[j + 1] = tmp;

}

}

}

t2 = clock();

return (t2 - t1);

}

friend void InVector(Vector &A){

for (int i = 0; i < A.get\_length(); i++){

A[i]= rand();

}

}

friend clock\_t InsertionSort(Vector& a)

{

clock\_t t1, t2;

t1 = clock();

T1 newElement;

int location;

for (int i = 1; i < a.get\_length(); i++)

{

newElement = a.get\_vector()[i];

location = i - 1;

while (location >= 0 && a.get\_vector()[location] > newElement)

{

a.get\_vector()[location + 1] = a.get\_vector()[location];

location = location - 1;

}

a.get\_vector()[location + 1] = newElement;

}

t2 = clock();

return t2 - t1;

}

friend clock\_t QuickSort(Vector& p, int first, int last)

{

clock\_t t1, t2, t3;

t1 = clock();

int i = first, j = last;

T1 tmp;

T1 x = p.get\_vector()[(first + last) / 2];

do {

while (p.get\_vector()[i] < x)

i++;

while (p.get\_vector()[j] > x)

j--;

if (i <= j)

{

if (i < j)

{

tmp = p.get\_vector()[i];

p.get\_vector()[i] = p.get\_vector()[j];

p.get\_vector()[j] = tmp;

}

i++;

j--;

}

} while (i <= j);

if (i < last)

t3 = QuickSort(p,i, last);

if (first < j)

t3 = QuickSort(p, first, j);

t2 = clock();

return (t2 - t1);

}

};

int main(){

Matrix<int>B;

Matrix<int>A(2, 2);

Matrix<int>C(A);

cout << C.get\_width() << " getter shiriny\n";

cout << C.get\_length() << " getter dliny\n";

cout << C.get\_array()[0][0] << " getter matrix\n";

B.set\_width\_length(2, 2);

cout << B.get\_width() << " " << B.get\_length() << "setter dliny i shiriny\n";

cin >> A >> B;

cout << "operator >> \n";

cout << B[0][0] << " operator [ ]\n";

C = A + B;

cout <<C<< " operator +\n";

C = A - B;

cout << C << "operator -\n";

cout << A\*B << " operator \*\n";

cout << (A == B) << " operator ==\n";

Vector<float>G;

Vector<float>H(5);

Vector<float>I(H);

cout << H << " operator <<\n";

cout << H.get\_vector()[0]<<" getter vector\n";

cout << H[0] << " operator [ ]\n";

G.set\_length(5);

cin >> G;

cin >> H;

cout << H / G << " operator /\n";

I = G+H;

cout << G\*H << " operator \*\n";

cout << I;

Vector<int>K(10000);

cout << BubbleSort(K) << endl;

cout << InsertionSort(K) << endl;

cout << QuickSort(K,0, K.get\_length() - 1)<<endl;

Matrix <int> P(1, 2);

P[0][0] = 1;

cout << P[0][0];

return 0;

}