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

Отчёт по лабораторной работе

Структуры хранения матриц специального вида

Выполнил:

студент института ИТММ гр. 381908-4

Иссаракура Н.А.

Проверил:

ассистент каф. МОСТ, ИИТММ

Лебедев И.Г.

Содержание

[Введение 3](#_Toc59367332)

[Постановка задачи 4](#_Toc59367333)

[Руководство пользователя 5](#_Toc59367334)

[Руководство программиста 6](#_Toc59367335)

[Описание алгоритмов 6](#_Toc59367336)

[Эксперименты 7](#_Toc59367337)

[Заключение 8](#_Toc59367338)

[Литература 9](#_Toc59367339)

[Приложения 10](#_Toc59367340)

Введение

Матрицы - один из важнейших объектов, имеющих прикладное значение. Матричные обозначения широко распространены в современной математике и её приложениях. Матрица – полезный аппарат для исследования многих задач теоретической и прикладной математики. Так, одной из важнейших является задача нахождения решения систем линейных алгебраических уравнений.

Следствием разнообразия областей применения матричного аппарата в современной науке является наличие в любом из больших математических программных комплексов (Mathcad, Mathematica, Derive, Mapple) подсистем, выполняющих операции над матрицами, а также существование специальных программных библиотек (ScalaPack, PlaPack), рассчитанных на обработку огромных (десятки и сотни тысяч строк) матриц, в том числе с использованием распределенных (параллельных) вычислений.

Помимо матриц общего вида, для которых наиболее естественной и наиболее часто используемой представляется программная реализация в виде двумерного массива, в математических приложениях выделяются различные матрицы специальных видов (треугольные, диагональные, …). Для таких матриц предпочтительно создание собственных способов хранения и обработки, учитывающих специфику их структуры, и потому более эффективных. Изучению некоторых из них посвящена данная работа.

# Постановка задачи

В рамках лабораторной работы ставится задача создания программных средств, поддерживающих эффективное хранение матриц специального вида (верхнетреугольных) и выполнение основных операций над ними:

* сложение/вычитание;
* копирование;
* сравнение.

В процессе выполнения лабораторной работы требуется использовать систему контроля версий [Git](https://git-scm.com/book/ru/v2) и фрэймворк для разработки автоматических тестов [Google Test](https://github.com/google/googletest).

Перед выполнением работы студенты получают данный проект-шаблон, содержащий следующее:

* Интерфейсы классов Вектор и Матрица (h-файл)
* Начальный набор готовых тестов для каждого из указанных классов.
* Набор заготовок тестов для каждого из указанных классов.
* Тестовый пример использования класса Матрица

Выполнение работы предполагает решение следующих задач:

1. Реализация методов шаблонного класса TVector согласно заданному интерфейсу.
2. Реализация методов шаблонного класса TMatrix согласно заданному интерфейсу.
3. Обеспечение работоспособности тестов и примера использования.
4. Реализация заготовок тестов, покрывающих все методы классов TVector и TMatrix.
5. Модификация примера использования в тестовое приложение, позволяющее задавать матрицы и осуществлять основные операции над ними.

# Руководство пользователя

Пользователю нужно запустить файл main.exe или Main (unix executable file если вы используете macOS).

Откроется консольное приложение для тестирования матриц.

Программа заполнит две матрица числами и выведет их в консоль, так же выведет результат сложения матриц.

Для повторного выполнения потребуется перезапустить программу.

# Руководство программиста

**Реализует операции:**

bool operato r== (const TMatrix& mt) const; //сравнение Матриц

TMatrix operator = (const TMatrix& mt); //присваивание Матриц

TMatrix operator+ (const TMatrix& mt); //сложение Матриц

TMatrix operator- (const TMatrix& mt); //вычитание Матриц

TMatrix operator\* (const TMatrix& mt); //умножение Матриц

Для структуры данных Вектор были реализованы следующие операции:

• вычисление длины;

• сравнение;

• прибавление/вычитание скаляра;

• умножение на скаляр;

• сложение/вычитание векторов;

• скалярное произведение векторов;

• создание копии.

Для структуры данных Матрица были реализованы следующие операции:

• сравнение:

• сложение/вычитание матриц:

• умножение матриц.

С учетом структуры данных целесообразной представляется следующая модульная структура программы:

• Vector.h, Vector.cpp – модуль, реализующий структуру данных Вектор;

• Matrix.h, Matrix.cpp – модуль, реализующий структуру данных Матрица;

• test\_matrix.cpp – модуль программы тестирования матриц.

• test\_vector.cpp – модуль программы тестирования векторов.

# Описание алгоритмов

Создание вектора:

* Инициализируем размер вектора;
* Заполняем вектор;

Создание матрицы:

* Инициализируем строки матрицы
* Заполняем строки элементами от 0 до кол-во векторов, которые понадобятся для создание матрицы;

# Эксперименты

Graphical user interface, text, application

Description automatically generated

Результат выполнения операции сложения:

Graphical user interface, text, application

Description automatically generated

Результат выполнения операции вычитания:

# Заключение

Успешно реализована и протестирована матрица верхнетреугольного вида и основные операции над ней. Отработаны навыки работы с шаблонами и с google tests.

В ходе выполнения работы были реализованы классы вектора и верхнетреугольной матрицы используя такие возможности как шаблонные классы и функции, перегрузка операций, а также наследование классов.

Были написаны тесты на базе **Google Test**, помогавшие с поиском ошибок в коде. В этом также помогала и sample-программа, редактирование которой помогало в более детальном поиске ошибок.

# Литература

1. Столлингс, В. Структурная организация и архитектура компьютерных систем, 5-е изд.: Пер. с англ. — М.: Издательский дом «Вильямс», 2002. — 896 с.: ил. — Парал. тит. англ.
2. Johnson M. Superscalar Microprocessor Design. — Englewood Cliff, New Jersey: Prentice Hall, 1991.
3. Касперски К. Техника оптимизации программ. Эффективное использование памяти. — СПб.: БХВ-Петербург, 2003. — 464 с.: ил.
4. Stone H. High performance Computer Architecture. — Reading, MA: Addison-Wesley, 1993.
5. Tullsen D.M., Eggers S.J. Effective Cache Prefetching on a Bus-Based Multiprocessor. — ACM Transactions on Computer Systems, pp. 57-88, Feb 1995.
6. Chandra D., Guo F., Kim S., Solihin Y. Predicting inter-thread cache contention on a chip multi-processor architecture. — Proceedings of the 11th International Symposium on High Performance Computer Architecture (HPCA), pp. 340–351, Feb 2005.
7. Press W., Teukolsky S., Vetterling W., Flannery B. Numerical Recipes in C. The Art of Scientific Computing. Second Edition. — Cambridge University Press, 1992.
8. Камаев А.М., Сиднев А.А., Сысоев А.В. Об одном подходе к анализу эффективности приложений // Труды 50-й научной конференции МФТИ «Современные проблемы фундаментальных и прикладных наук»: Часть I. Радиотехника и кибернетика. - М.: МФТИ, 2007.
9. Debugging and performance monitoring. Intel® 64 and IA-32 Architectures Software Developer’s Manual. Volume 3B: System Programming Guide, Part 2. May 2007. — [http://www.intel.com/products/processor/manuals/]
10. Юнаковский А.Д. Начала вычислительных методов для физиков. – Н. Новгород: ИПФ РАН, 2007.

# Приложения

Graphical user interface, text, application, Word

Description automatically generated

Text

Description automatically generated with medium confidence

#pragma once

#include "MyVector.h"

template<class T>

class TMatrix : public Vector<Vector<T> >

{

int size;

public:

TMatrix(int \_size = 0);

TMatrix(const TMatrix& A);

TMatrix(const Vector<Vector<T> >& A);

~TMatrix();

T& operator()(int row, int col) const;

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

TMatrix operator+(const TMatrix& mt) const;

TMatrix operator-(const TMatrix& mt) const;

TMatrix operator\*(const TMatrix& mt) const;

bool operator==(const TMatrix& mt) const;

bool operator!=(const TMatrix& mt) const;

friend ostream& operator<<(ostream& out,const TMatrix& mt) {

for (int i = 0; i < mt.length; i++) {

out << mt.x[i] << "\n";

}

return out;

}

friend istream& operator>>(istream& in, TMatrix& mt) {

for (int i = 0; i < mt.Lenngth(); i++)

in >> mt.pVector[i];

return in;

}

};

template<class T>

inline TMatrix<T>::TMatrix(int \_size) : Vector<Vector<T> >(\_size)

{

if (\_size < 0)

throw new std::exception();

this->size = \_size;

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

this->x[i] = Vector<T>(\_size - i);

}

template<class T>

inline TMatrix<T>::TMatrix(const TMatrix& A) : Vector<Vector<T> >(A)

{

this->size = A.size;

}

template<class T>

inline TMatrix<T>::TMatrix(const Vector<Vector<T> >& A) : Vector<Vector<T> >(A)

{

}

template<class T>

inline TMatrix<T>::~TMatrix()

{

}

template<class T>

inline T& TMatrix<T>::operator()(int row, int col) const

{

if (row < 0 || row >= this->size)

throw new std::exception();

if (col < 0 || col >= this->size)

throw new std::exception();

return this->x[row][col - row];

}

template<class T>

TMatrix<T>& TMatrix<T>::operator=(const TMatrix<T>& mt)

{

if (this == &mt)

return \*this;

if (this->x != NULL)

delete[] this->x;

Vector<Vector<T> >:: operator=(mt);

return \*this;

}

template<class T>

TMatrix<T> TMatrix<T>::operator+(const TMatrix<T>& mt) const

{

TMatrix<T> tmp(\*this);

if (this->length != mt.length)

throw new exception();

for (int i = 0; i < tmp.length; i++)

tmp.x[i] = tmp.x[i] + mt.x[i];

return tmp;

}

template<class T>

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

{

TMatrix<T> tmp(\*this);

if (this->length != mt.length)

throw new exception();

for (int i = 0; i < tmp.length; i++)

tmp.x[i] = tmp.x[i] - mt.x[i];

return tmp;

}

template<class T>

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

{

if (this->size != mt.size)

{

throw new std::exception();

}

TMatrix<T> m(\*this);

TMatrix<T> res(this->size);

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

{

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

{

res.x[i][j] = 0;

for (int k = 0; k < this->size; k++)

res.x[i][j] += (m.x[i][k] \* mt.x[k][j]);

}

}

return res;

}

template<class T>

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

{

if (this->size != mt.size)

return false;

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

if (this->x[i] != mt.x[i])

return false;

return true;

}

template<class T>

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

{

return !(\*this == mt);

}

#ifndef \_MY\_VECTOR\_

#define \_MY\_VECTOR\_

#include <iostream>

using namespace std;

template <class T>

class Vector

{

protected:

int length;

T\* x;

public:

Vector<T>\* vec;

Vector();

Vector(int \_size);

Vector(int rowsCount, T\* \_v);

Vector(int rowsCount, T \_v);

Vector(const Vector<T>& \_v);

virtual ~Vector();

bool operator==(const Vector<T>& \_v);

bool operator!=(const Vector<T>& \_v);

Vector<T> operator +(const Vector<T>& \_v);

Vector<T> operator -(Vector<T>& \_v);

Vector<T> operator \*(Vector<T>& \_v);

Vector<T> operator /(Vector<T>& \_v);

Vector<T>& operator =(const Vector<T>& \_v);

T& operator[] (const int index);

Vector<T>& operator ++();

Vector<T>& operator --();

Vector<T>& operator +=(Vector<T>& \_v);

Vector<T>& operator -=(Vector<T>& \_v);

template <class T1>

friend ostream& operator<< (ostream& ostr, const Vector<T1> &A);

template <class T1>

friend istream& operator >> (istream& istr, Vector<T1> &A);

int Length();

};

template <class T1>

ostream& operator<< (ostream& ostr, const Vector<T1> &A) {

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

ostr << A.x[i] << " ";

}

return ostr;

}

template <class T1>

istream& operator >> (istream& istr, Vector<T1> &A) {

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

istr >> A.x[i];

}

return istr;

}

#define MIN(a,b)(a>b?b:a)

#define MAX(a,b)(a>b?a:b)

template <class T>

Vector<T>::Vector()

{

length = 0;

x = 0;

}

template <class T>

Vector<T>::Vector(int \_size)

{

if (\_size < 0)

throw new exception();

length = \_size;

x = new T[length];

}

template <class T>

Vector<T>::Vector(int rowsCount, T\* \_v)

{

length = rowsCount;

///x = \_v;

x = new T [length];

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

x[i] = \_v[i];

}

template <class T>

Vector<T>::Vector(int rowsCount, T \_v)

{

length = rowsCount;

x = new T [length];

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

x[i] = \_v;

}

template <class T>

Vector<T>::Vector(const Vector<T>& \_v)

{

length = \_v.length;

x = new T [length];

for (int i = 0; i < length;i = i + 1)

x[i] = \_v.x[i];

}

template <class T>

Vector<T>::~Vector()

{

length = 0;

if (x != 0)

delete[] x;

x = 0;

}

template<class T>

inline bool Vector<T>::operator==(const Vector<T>& \_v)

{

if (this->length != \_v.length)

return false;

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

if (x[i] != \_v.x[i])

return false;

return true;

}

template<class T>

inline bool Vector<T>::operator!=(const Vector<T>& \_v)

{

return !(\*this==\_v);

}

template <class T>

Vector<T> Vector<T>::operator +(const Vector<T>& \_v)

{

Vector<T> res;

res.length = MIN(length, \_v.length);

res.x = new T [res.length];

for (int i = 0; i < res.length; i++)

{

res.x[i] = x[i] + \_v.x[i];

}

return res;

}

template <class T>

Vector<T> Vector<T>::operator -(Vector<T>& \_v)

{

Vector<T> res;

res.length = MIN(length, \_v.length);

res.x = new T [res.length];

for (int i = 0; i < res.length; i++)

{

res.x[i] = x[i] - \_v.x[i];

}

return res;

}

template <class T>

Vector<T> Vector<T>::operator \*(Vector<T>& \_v)

{

Vector<T> res;

res.length = MIN(length, \_v.length);

res.x = new T [res.length];

for (int i = 0; i < res.length; i++)

{

res.x[i] = x[i] \* \_v.x[i];

}

return res;

}

template <class T>

Vector<T> Vector<T>::operator /(Vector<T>& \_v)

{

Vector<T> res;

res.length = MIN(length, \_v.length);

res.x = new T [res.length];

for (int i = 0; i < res.length; i++)

{

res.x[i] = x[i] / \_v.x[i];

}

return res;

}

template <class T>

Vector<T>& Vector<T>::operator =(const Vector<T>& \_v)

{

if (this == &\_v)

return \*this;

length = \_v.length;

x = new T [length];

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

x[i] = \_v.x[i];

return \*this;

}

template <class T>

T& Vector<T>::operator[] (const int index)

{

if ((index >= 0) && (index < length))

return x[index];

return x[0];

}

template <class T>

Vector<T>& Vector<T>::operator ++()

{

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

x[i]++;

return \*this;

}

template <class T>

Vector<T>& Vector<T>::operator --()

{

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

x[i]--;

return \*this;

}

template <class T>

Vector<T>& Vector<T>::operator +=(Vector<T>& \_v)

{

length = MIN(length, \_v.length);

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

{

x[i] += \_v.x[i];

}

return \*this;

}

template <class T>

Vector<T>& Vector<T>::operator -=(Vector<T>& \_v)

{

length = MIN(length, \_v.length);

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

{

x[i] -= \_v.x[i];

}

return \*this;

}

template <class T>

int Vector<T>::Length()

{

return length;

}

#endif