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 ф-та ПРИН гр. 381908-4

Кургузиков Кирилл Александрович

Проверил:

ассистент каф. МОСТ, ИИТММ

Лебедев И.Г

Нижний Новгород

2020 г.

Содержание

[Введение 2](#_Toc53768280)

[Постановка задачи 5](#_Toc53768281)

[Руководство программиста 6](#_Toc53768282)

[Описание структуры программы 6](#_Toc53768283)

[Описание алгоритмов 7](#_Toc53768284)

[Эксперименты 9](#_Toc53768285)

[Заключение 11](#_Toc53768286)

[Литература 12](#_Toc53768287)

[Приложения 13](#_Toc53768288)

# Введение

Матрица — математический объект, записываемый в виде прямоугольной таблицы элементов кольца или поля, которая представляет собой совокупность строк и столбцов, на пересечении которых находятся её элементы. Количество строк и столбцов задает размер матрицы. Хотя исторически рассматривались, например, треугольные матрицы, в настоящее время говорят исключительно о матрицах прямоугольной формы, так как они являются наиболее удобными и общими.

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.

В программировании матрица – это структура данных, хранящая набор значений (элементов массива), идентифицируемых по индексу или набору индексов, принимающих целые (или приводимые к целым) значения из некоторого заданного непрерывного диапазона.

**Руководство пользователя**

Пользователю нужно запустить файл main.exe.

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.

# Постановка задачи

В данной лабораторной работе необходимо разработать программу, которая реализует эффективное хранение матриц специального вида, и позволяет производить основные операции над матрицами:

* сложение;
* вычитание;
* умножение;
* копирование;
* сравнение.

В реализации нужно использовать класс TVector и класс TMatrix.

# Руководство программиста

## Описание структуры программы

#include "MyVector.h"– подключение файла MyVector.h, который описывает работу с векторами.

#include "Matrix.h" – подключение файла Matrix.h, который описывает работу с множествами, на основе битовых полей.

В функции «main» происходит создание трех матриц, заполнения двух из них значениями. В третью матрицу записывается результат сложения первых двух. После всего выводится результат в консоль.

Код классов в приложении.

Описание структур данных

Программа состоит из нескольких основных классов – TVector, TMatrix.

**TVector.**

Реализованы конструктор по умолчанию, конструктор с параметром, конструктор копирования, деструктор.

Содержит поля:

* ValType\* pVector для предоставления памяти вектору,
* Size – размер вектора.

Реализует методы:

* GetSize – получение размера вектора,

Реализует операции:

* operator== - оператор сравнения,
* operator[] – доступ к элементам вектора,
* operator= - оператор присваивания,
* operator+ - прибавить скаляр,
* operator- - вычесть скаляр,
* operator\* - умножить на скаляр
* operator/ - делить на скаляр
* operator+ - сложение векторов
* operator- - вычитание векторов
* operator\* - умножение векторов
* &operator>> -оператор ввода,
* &operator<< - оператор вывода.

**TMatrix.h**

Реализованы конструктор с параметром, конструктор копирования, конструктор преобразования типа, деструктор.

Содержит поля:

* mSize – размер матрицы.

Реализует методы:

* GetSize – получение размера.

Реализует операции:

* operator== - оператор сравнения,
* operator= - оператор присваивания,
* operator+ - оператор сложения,
* operator- - оператор разности,
* operator\* - оператор умножения,
* &operator>> -оператор ввода,
* &operator<< - оператор вывода.

## Описание алгоритмов

Создание вектора:

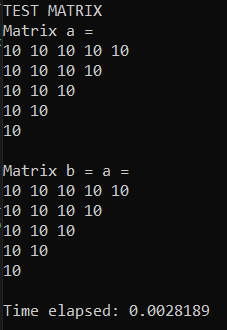
* Проверяем значение размера на допуск,
* Выделяем память,
* Заполняем вектор.

Создание матрицы:

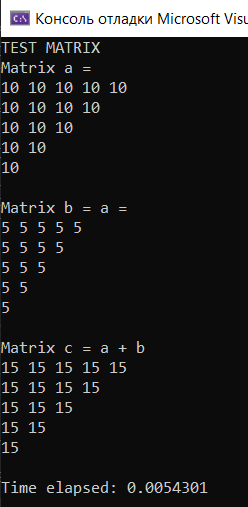
* Проверяем значение размера на допуск,
* Создаем вектор,
* Инициализируем строки матрицы,
* Заполняем мятрицу.

# Эксперименты

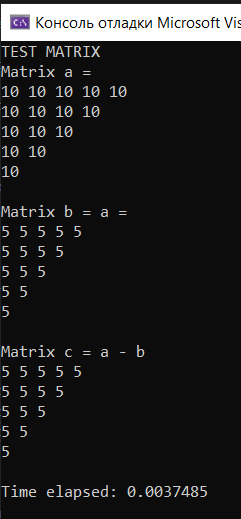
Результат выполнения операции присваивания:



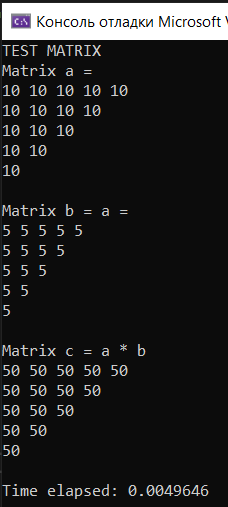
Результат выполнения операции сложения:



Результат выполнения операции вычитания:



Результат выполнения операции умножения:



# Заключение

В ходе выполнения данной лабораторной работы был реализован класс TMatrix на основе класса TVector. Реализованы основные операции над матрицами.

# 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

Код TVector.h

#ifndef \_MY\_VECTOR\_

#define \_MY\_VECTOR\_

#include <iostream>

using namespace std;

const int MAX\_VECTOR\_SIZE = 100000;

//Шаблон вектора

template <class ValType>

class TVector

{

protected:

ValType\* pVector;

int Size;

int StartIndex;

public:

TVector(int s = 10, int si = 0);

TVector(const TVector& v); // конструктор копирования

virtual ~TVector();

int GetSize() { return Size; } // размер вектора

int GetStartIndex() { return StartIndex; } //Индекс первого элемента

ValType& operator[](int pos); // доступ

bool operator==(const TVector& v) const; // сравнение

TVector& operator=(const TVector& v); // присваивание

// скалярные операции

TVector operator+(const ValType& val); // прибавить скаляр

TVector operator-(const ValType& val); // вычесть скаляр

TVector operator\*(const ValType& val); // умножить на скаляр

TVector operator/(const ValType& val); //делить на скаляр

// векторные операции

TVector operator+(const TVector& v); // сложение

TVector operator-(const TVector& v); // вычитание

TVector operator\*(const TVector& v); // скалярное произведение

// ввод-вывод

friend istream& operator>>(istream& in, TVector& v)

{

for (int i = 0; i < v.Size; i++)

in >> v.pVector[i];

return in;

}

friend ostream& operator<<(ostream& out, const TVector& v)

{

for (int i = 0; i < v.Size; i++)

out << v.pVector[i] << ' ';

return out;

}

};

template <class ValType>

TVector<ValType>::TVector(int s, int si)

{

if (s < 0 || s > MAX\_VECTOR\_SIZE || si < 0)

{

throw "Error";

}

pVector = new ValType[s];

Size = s;

StartIndex = si;

}

template <class ValType> //конструктор копирования

TVector<ValType>::TVector(const TVector<ValType>& v)

{

pVector = new ValType[v.Size];

Size = v.Size;

StartIndex = v.StartIndex;

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

{

pVector[i] = v.pVector[i];

}

}

template <class ValType>

TVector<ValType>::~TVector()

{

if (pVector != 0)

{

delete[] pVector;

}

pVector = 0;

Size = 0;

StartIndex = 0;

}

template <class ValType> // доступ

ValType& TVector<ValType>::operator[](int index)

{

if (index < 0 || index > Size)

{

throw "Error";

}

return pVector[index - StartIndex];

}

template <class ValType> // сравнение

bool TVector<ValType>::operator==(const TVector& v) const

{

bool res = true;

if (Size != v.Size)

{

res = false;

}

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

{

if (pVector[i] != v.pVector[i])

{

res = false;

}

}

return res;

}

template <class ValType> // присваивание

TVector<ValType>& TVector<ValType>::operator=(const TVector& v)

{

if (this == &v)

{

return \*this;

}

if (Size != v.Size)

{

if (pVector != NULL)

{

delete[] pVector;

}

pVector = new ValType[v.Size];

}

Size = v.Size;

StartIndex = v.StartIndex;

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

{

pVector[i] = v.pVector[i];

}

return \*this;

}

template <class ValType> // прибавить скаляр

TVector<ValType> TVector<ValType>::operator+(const ValType& val)

{

TVector<ValType> res;

res.Size = Size;

res.pVector = new ValType[res.Size];

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

{

res.pVector[i] = pVector[i] + val;

}

return res;

}

template <class ValType> // вычесть скаляр

TVector<ValType> TVector<ValType>::operator-(const ValType& val)

{

TVector<ValType> res;

res.Size = Size;

res.pVector = new ValType[res.Size];

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

{

res.pVector[i] = pVector[i] - val;

}

return res;

}

template <class ValType> // умножить на скаляр

TVector<ValType> TVector<ValType>::operator\*(const ValType& val)

{

TVector<ValType> res;

res.Size = Size;

res.pVector = new ValType[res.Size];

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

{

res.pVector[i] = pVector[i] \* val;

}

return res;

}

template<class ValType>

inline TVector<ValType> TVector<ValType>::operator/(const ValType& val)

{

TVector<ValType> res;

res.Size = Size;

res.pVector = new ValType[res.Size];

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

{

res.pVector[i] = pVector[i] / val;

}

return res;

}

template <class ValType> // сложение

TVector<ValType> TVector<ValType>::operator+(const TVector<ValType>& v)

{

if (v.Size != Size)

{

throw "Error";

}

TVector<ValType> res;

res.Size = Size;

res.pVector = new ValType[res.Size];

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

{

res.pVector[i] = pVector[i] + v.pVector[i];

}

return res;

}

template <class ValType> // вычитание

TVector<ValType> TVector<ValType>::operator-(const TVector<ValType>& v)

{

if (v.Size != Size)

{

throw "Error";

}

TVector<ValType> res;

res.Size = Size;

res.pVector = new ValType[res.Size];

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

{

res.pVector[i] = pVector[i] - v.pVector[i];

}

return res;

}

template <class ValType> // скалярное произведение

TVector<ValType> TVector<ValType> ::operator\*(const TVector<ValType>& v)

{

if (v.Size != Size)

{

throw "Error";

}

TVector<ValType> res;

res.Size = Size;

res.pVector = new ValType[res.Size];

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

{

res.pVector[i] = pVector[i] \* v.pVector[i];

}

return res;

}

#endif

Код TMatrix.h

#ifndef \_MATRIX\_

#define \_MATRIX\_

#include "MyVector.h"

#include <fstream>

const int MAX\_MATRIX\_SIZE = 100000;

// Верхнетреугольная матрица

template <class ValType>

class TMatrix : public TVector<TVector<ValType> >

{

private:

int mSize;

public:

TMatrix(int s = 10);

TMatrix(const TMatrix& mt); // копирование

TMatrix(const TVector<TVector<ValType> >& mt); // преобразование типа

~TMatrix();

int GetSize() { return mSize; }; //Получение размера

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

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

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

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

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

// ввод / вывод

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

{

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

{

in >> mt.pVector[i];

}

return in;

}

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

{

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

{

out << mt.pVector[i] << endl;

}

return out;

}

};

template<class ValType>

inline TMatrix<ValType>::TMatrix(int s) : TVector<TVector <ValType> >(s)

{

if (s < 0 || s > MAX\_MATRIX\_SIZE)

{

throw "Error";

}

mSize = s;

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

{

pVector[i] = TVector<ValType>(s - i);

}

}

template <class ValType> // конструктор копирования

inline TMatrix<ValType>::TMatrix(const TMatrix<ValType>& mt) : TVector<TVector<ValType> >(mt)

{

mSize = mt.mSize;

}

template <class ValType> // конструктор преобразования типа

inline TMatrix<ValType>::TMatrix(const TVector<TVector<ValType> >& mt) : TVector<TVector<ValType> >(mt)

{

}

template<class ValType>

inline TMatrix<ValType>::~TMatrix()

{

if (mSize != 0)

{

mSize = NULL;

}

}

template <class ValType> // сравнение

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

{

bool res = true;

int S = this->Size;

if (S != mt.Size)

{

res = false;

}

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

{

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

{

res = true;

}

else res = false;

}

return res;

}

template <class ValType> // присваивание

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

{

if (this != &mt)

{

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

{

if (this->pVector != NULL)

{

delete[] this->pVector;

}

this->pVector = new TVector<ValType>[mt.Size];

}

this->Size = mt.Size;

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

{

this->pVector[i] = mt.pVector[i];

}

}

return \*this;

}

template <class ValType> // сложение

inline TMatrix<ValType> TMatrix<ValType>::operator+(const TMatrix<ValType>& mt)

{

if (this->GetSize() != mt.Size)

{

throw "Error";

}

TMatrix<ValType> tmp(\*this);

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

{

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

}

return tmp;

}

template <class ValType> // вычитание

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

{

if (this->GetSize() != mt.Size)

{

throw "Error";

}

TMatrix<ValType> tmp(\*this);

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

{

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

}

return tmp;

}

template<class ValType>

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

{

if (this->GetSize() != mt.Size)

{

throw "Error";

}

TMatrix<ValType> tmp(\*this);

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

{

tmp.pVector[i] = tmp.pVector[i] \* mt.pVector[i];

}

return tmp;

}

#endif