

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
Галиндо Хавьер Э.

Проверил:
ассистент каф. МОСТ, ИИТММ

Лебедев И.Г.

Нижний Новгород
2020 г.

Содержание

Введение	3
Постановка задачи	4
Руководство пользователя	4
Руководство программиста	6
Описание структуры программы.....	6
Описание алгоритмов	6
Эксперименты	8
Заключение	9
Литература.....	10
Приложение	11

Введение

В программировании матрица - это набор чисел, расположенных в строках и столбцах.

Числа в матрице могут представлять данные или математические уравнения. Они используются как способ быстрого приближения более сложных вычислений.

Поскольку математическая концепция матрицы может быть представлена в виде двумерной сетки, двумерные массивы также иногда называют матрицами. В некоторых случаях термин “вектор” используется в вычислениях для обозначения массива.

Актуальностью данной работы является то, что - это способ хранения данных в организованной форме в виде строк и столбцов. Обычно матрицы используются в компьютерной графике для проецирования трехмерного пространства на двумерный экран. Матрицы в виде массивов используются для хранения данных в организованной форме.

Целью данной работы является - разработка структуры программных инструментов, которые поддерживают быстрое и эффективное хранение треугольных матриц и выполняют базовые операции, такие как сложение / вычитание, умножение, копирование и сравнение.

Практическая значимость данной работы - создать набор программных инструментов, которые поддерживают быстрое и эффективное хранение треугольных матриц, выполняют базовые операции, и применить их непосредственно на практике.

Постановка задачи

Задача работы — создать набор программных инструментов, которые поддерживают быстрое и эффективное хранение треугольных матриц и выполняют базовые операции, такие как сложение / вычитание, умножение, копирование и сравнение.

Программа будет содержать:

- векторный класс
- матричный класс
- тестовое приложение, позволяющее определять матрицы и выполнять с ними базовые операции.

Программное обеспечение будет использовать систему контроля версий Git и платформу для разработки автоматизированных тестов Google Test.

Руководство пользователя

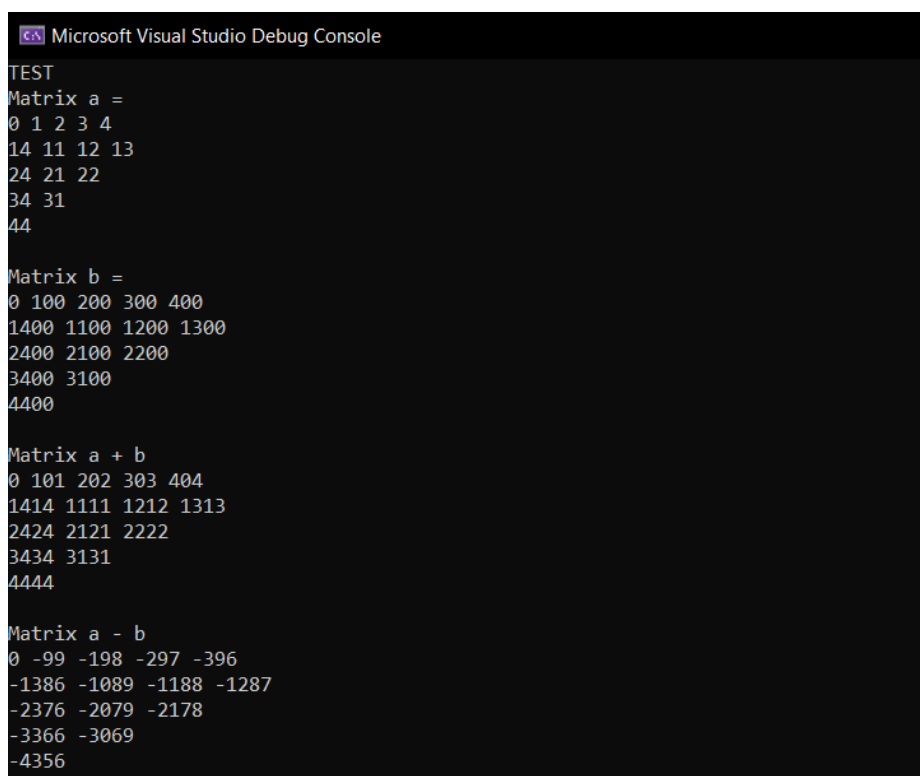
Для начала работы пользователю нужно запустить файл Main.exe.

Далее откроется консольное приложение для тестирования матриц.

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

Для повторного выполнения вычислений потребуется перезапустить программу.

Результаты матрицы и описание процессов представлены на рисунке 1.

A screenshot of the Microsoft Visual Studio Debug Console window. The window has a title bar with the Visual Studio logo and the text "Microsoft Visual Studio Debug Console". The console output is as follows:

```
TEST
Matrix a =
0 1 2 3 4
14 11 12 13
24 21 22
34 31
44

Matrix b =
0 100 200 300 400
1400 1100 1200 1300
2400 2100 2200
3400 3100
4400

Matrix a + b
0 101 202 303 404
1414 1111 1212 1313
2424 2121 2222
3434 3131
4444

Matrix a - b
0 -99 -198 -297 -396
-1386 -1089 -1188 -1287
-2376 -2079 -2178
-3366 -3069
-4356
```

Рисунок 1 – Результат работы матрицы

Руководство программиста

Описание структуры программы

Программа состоит из следующих файлов:

- MyVector.h, MyVector: класс векторов и методы для реализации класса векторов
- Matrix.h, Matrix.cpp: класс матриц и методы для реализации класса матриц.
- main.cpp: тестовый файл.
- sample_matrix.cpp: модуль программы тестирования, с которым работает пользователь, в котором проводятся эксперименты.

В программе реализованы классы, соответствующие функционированию программы.

Описание алгоритмов

реализация методов шаблонного класса TVector:

класс MyVector

- Метод “GetValue”

Получение значения указанного элемента в векторе. Это происходит путём нахождения разницы указанной позиции и начального индекса в векторе.

```
Тип& TVector<Тип>:: GetValue(цел pos) {  
    если (pos < 0)  
        throw логическое исключение("Input error: index cannot be negative value.");  
    если (startIndex > pos)  
        throw исключение выхода из диапазона("Input error: index is out of range, less then  
start index.");  
    если (pos >= size + startIndex)  
        throw исключение выхода из диапазона ("Input error: index is out of range, more then  
size.");  
    вернуть pVector[pos - startIndex];  
}
```

- Оператор “Operator+”

Сложение происходит путём прибавления аргумента к значению всех элементов вектора. Возвращает вектор.

```
TVector<символ> TVector<символ>::Оператор operator+(const ValType &val) {  
    TVector<символ> tmp(size, startIndex);  
    цел i;  
    цикл (i = 0; i < size; ++i)  
        tmp.pVector[i] = pVector[i] + val;  
    вернуть tmp;  
}
```

Класс Matrix

- Оператор “Operator=”

Происходит присваивание вектора. Если размеры и стартовые индексы двух векторов не соответствуют происходит их присваивания этих переменных. Если равны, то сразу происходит присваивание значений одного вектора другому.

```
TMatrix<символ>& TMatrix<символ>::Оператор operator= (const TMatrix<символ>  
&mt) {  
    если (не(*this == mt)) {  
        цел i;  
        если (не(this->size == mt.size)) {  
            (*this).size = mt.size;  
            (*this).startIndex = mt.startIndex;  
        }  
        цикл (i = 0; i < this->size; ++i)  
            (*this).pVector[i] = mt.pVector[i];  
    } вернуть *this;  
}
```

Эксперименты

Результат выполнения операции присваивания:

```
TEST MATRIX
Matrix a =
10 10 10 10 10
10 10 10 10
10 10 10
10 10
10

Matrix b = a =
10 10 10 10 10
10 10 10 10
10 10 10
10 10
10

Time elapsed: 0.0028189
```

Результат выполнения операции присваивания:

```
TEST MATRIX
Matrix a =
10 10 10 10 10
10 10 10 10
10 10 10
10 10
10

Matrix b = a =
5 5 5 5 5
5 5 5 5
5 5 5
5 5
5

Matrix c = a + b
15 15 15 15 15
15 15 15 15
15 15 15
15 15
15

Time elapsed: 0.0054301
```


Заключение

Подытожим что целью данной работы являлось - разработка структуры программных инструментов, которые поддерживают быстрое и эффективное хранение треугольных матриц и выполняют базовые операции, такие как сложение / вычитание, умножение, копирование и сравнение.

Как видно из структуры данной работы результатом стала - структура данных для хранения вектора и верхней треугольной матрицы.

Классы шаблонов были протестированы с помощью Google Tests, и были проведены эксперименты для сравнения теоретической и практической сложности выполнения операций над методом класса, который был разработан с помощью имеющихся инструментов.

А также использована практическая значимость данной работы - создать набор программных инструментов, которые поддерживают быстрое и эффективное хранение треугольных матриц, выполняют базовые операции, и применить их непосредственно на практике.

Литература

1. Круз Р.Л. Структуры данных и проектирование программ. – М.: Бином. Лаборатория знаний, 2014.
2. Топп У., Форд У. Структуры данных в C++.- М.: Бином, 1999
3. Кормен Т., Лейзерсон Ч., Ривест Р. Алгоритмы: построение и анализ.- М.: МЦМО, 1999.
4. Ахо Альфред В., Хопкрофт Джон, Ульман Джеффри Д. Структуры данных и алгоритмы- М.: Издательский дом Вильямс, 2000.
5. Гергель В.П. и др. Методы программирования. Учебное пособие. Н.Новгород: ННГУ, 2016.
6. Столлинс, В. Структурная организация и архитектура компьютерных систем, 5-е изд.: Пер. с англ. — М.: Издательский дом «Вильямс», 2002. — 896 с.: ил. — Парал. тит. англ.
7. Страуструп Б. Язык программирования C++. – М.: Бином, 2001

Приложение

TVector.h

```
#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

```

Matrix.h

```

#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);
}

```

Main.cpp

```

#include <iostream>

#include "MyVector.h"
#include "Matrix.h"

int main()
{
    TMatrix<int> a(5), b(5);

```

```

cout << "TEST" << endl;
for (int i = 0; i < 5; i++)
    for (int j = 0; j < 5; j++)
    {
        a[i][j] = i * 10 + j;
        b[i][j] = (i * 10 + j) * 100;
    }
TMatrix<int> c = a;
cout << "Matrix a = " << endl << a << endl;
cout << "Matrix b = " << endl << b << endl;
cout << "Matrix a + b " << endl << a + b << endl;
cout << "Matrix a - b " << endl << a - b << endl;
cout << c;
}

```