Федеральное агентство по образованию Российской Федерации
Государственное образовательное учреждение
высшего профессионального образования
Нижегородский государственный университет им. Н.И. Лобачевского

Факультет вычислительной математики и кибернетики

Отчёт по лабораторной работе

Отчёт матрицы

Выполнил:
студент ф-та ИИТММ гр. 381908-01

Козел С. А.

Проверил:
ассистент каф. МОСТ, ИИТММ

Лебедев И.Г.

Нижний Новгород
2020 г.

Содержание

Введение

Матрица – математический объект, записываемый в виде прямоугольной таблицы элементов, которые представляет собой совокупность строк и столбцов, на пересечении которых находятся его элементы. Количество строк и столбцов задает размер матрицы.

Для матриц существует множество видов представления (некоторые из них):

1. Треугольная матрица
2. Ленточная матрица
3. Диагональная

Постановка задачи

Цель данной работы - разработка структуры данных для хранения матриц с использованием векторов. Реализация треугольных, ленточных и разрежённых матриц.

Выполнение работы предполагает решение следующих задач:

1. Реализация класса вектора TVector.
2. Реализация класса треугольных матриц TMatrix и LowerMatrix.
3. Реализация класса ленточных матриц TapeMatrix
4. Реализация разряжённых матриц RazrMatrix
5. Реализация методов для ввода/вывода матриц в файл
6. Публикация исходных кодов в личном репозиторий на GitHub.

Руководство пользователя

Пользователю нужно запустить файл Matrix.exe.

Откроется консольное приложение для тестирования матриц.

Программа покажет функциональность каждой функции по средствам вывода данных в консоль.

Руководство программиста

Описание структуры программы

Программа состоит из следующих модулей:

- Приложение Matrix
- Статическая библиотека UMatrix:
  - UMatrix.h – описание класса вектора и различных матриц
- Приложение main:
  - main.cpp – тестирование работы программы

Описание структур данных

Класс TVector:

```
1. TVector(int s = 10, int si = 0) - конструктор по умолчанию;
2. TVector(const TVector& v) - конструктор копирования;
3. ~TVector() - диструктор;
4. int GetSize() - получить размер вектор;
5. int GetStartIndex() - получить индекс стартового элемента;
6. ValType& operator[](int pos) - перегрузка скобок для доступа;
7. bool operator==(const TVector& v) const – перегрузка сравнения векторов;
8. bool operator!=(const TVector& v) const – перегрузка неравенства векторов;
9. TVector& operator=(const TVector& v) – перегрузка операции присваивания;
10. TVector  operator+(const ValType& val) - прибавить скаляр;
11. TVector  operator-(const ValType& val) - вычесть скаляр;
12. TVector  operator*(const ValType& val) - умножить на скаляр;
13. TVector  operator+(const TVector& v) - сложение векторов;
14. TVector  operator-(const TVector& v) - вычитание векторов;
15. ValType  operator*(const TVector& v) - скалярное произведение векторов;
16. friend istream& operator>>(istream& in, TVector& v) - перегрузка ввода СД;
17. friend ostream& operator<<(ostream& out, const TVector& v) - перегрузка вывода СД;
```

Класс `TMatrix, LowerMatrix, RazrMatrix, TapeMatrix:`

```
1. TMatrix(const TMatrix& mt) - конструктор копирования;
2. TMatrix(const TVector<TVector<ValType>>& mt) - конструктор преобразование типа;
3. bool operator==(const TMatrix& mt) const - сравнение матриц;
4. bool operator!=(const TMatrix& mt) const - неравенство матриц;
5. TMatrix& operator= (const TMatrix& mt) - присваивание матриц;
6. TMatrix  operator+ (const TMatrix& mt) - сложение матриц;
7. TMatrix  operator- (const TMatrix& mt) - вычитание матриц;
8. TMatrix  operator* (const TMatrix& mt) - перемножение матриц;
9. TMatrix  operator+ (const ValType& num) - прибавить скаляр;
10. TMatrix  operator- (const ValType& num) - вычесть скаляр;
11. TMatrix  operator* (const ValType& num) - умножить на скаляр;
```

Описание алгоритмов

Принцип работы вектора.

Вектор - это структура данных, которая уже является моделью динамического массива. Вектор хранит в себе массив элементов, размер выделенной памяти, индекс стартового элемента вектора. Принцип работы по принципу массива, вставка элементов осуществляется по индексу, обращение к вектору происходит с помощью квадратных скобок.

Принцип работы матриц.

Матрица наследуется от класса вектора, каждая матрица представляет из себя список векторов, каждый вектор является рядом в матрице. При инициализации выделяется память для векторов(рядов).

Эксперименты

```
Тестирование программ поддержки представления треугольных матриц
Matrix a =
0 1 2 3 4
11 12 13 14
22 23 24
33 34
44

Matrix b =
0 100 200 300 400
1100 1200 1300 1400
2200 2300 2400
3300 3400
4400

Matrix c = a + b
0 101 202 303 404
1111 1212 1313 1414
2222 2323 2424
3333 3434
4444

Matrix c = a * b
0 1100 5600 15800 34000
12100 39600 84800 150000
48400 126500 236600
108900 261800
193600

Matrix c = a * 5
0 5 10 15 20
55 60 65 70
110 115 120
165 170
220

Matrix c = a + 5
5 6 7 8 9
16 17 18 19
27 28 29
38 39
49

Matrix c = a - 5
-5 -4 -3 -2 -1
6 7 8 9
17 18 19
28 29
39

1
3 4
5 6 7
7 8 9 10
9 10 11 12 13

Inpud data: 1 0 0 2 2 1 2 0 0 0 0 1 1 1 1 2 1 0 2 2 0 2 1 1 1

Razryazgennie matrix:
1 2 2 1 2 1 1 1 1 2 1 2 2 2 1 1 1
0 3 4 0 1 1 2 3 4 0 1 3 4 1 2 3 4
0 3 5 9 13 17

Klassic vid matrix:
1 2 2
1 2
1 1 1 1
2 1 2 2
2 1 1 1

Razryazgennie matrix r = a + a:
2 4 4
2 4
2 2 2 2
4 2 4 4
4 2 2 2

Razryazgennie matrix r = a * a:
24 48 48
24 48
24 24 24 24
48 24 48 48
48 24 24 24

Razryazgennie matrix r = a * 5:
5 10 10
5 10
5 5 5 5
10 5 10 10
10 5 5 5

Tape Matrix:
0 1
0 1 2
4 4 1
3 2 0
4 4 0
1 4
Tape matrix t = tape + tape:
0 2
0 2 4
8 8 2
6 4 0
8 8 0
2 8
Tape matrix t = tape + 5:
5 6
5 6 7
9 9 6
8 7 5
9 9 5
6 9
After read file m =
0 1 2 3 4
11 12 13 14
22 23 24
33 34
44
```

Заключение

При выполнении данной работы мною была полностью изучена и успешно реализована структура данных матрица с использованием векторов на массиве. Данная работа несёт в себе важный опыт для любого программиста, ведь вектора и матрицы самые популярные структуры данных.

Литература

1. https://studopedia.su/10_47593_ponyatie-lenti-matritsi-lentochniy-metod-hraneniya-razrezhennoy-matritsi.html
2. http://num-anal.srcc.msu.ru/lib_na/org/org_matr.htm
3. https://ru.wikipedia.org/wiki/Матрица_(математика)

Приложения

Приложение 1

UMatrix.h

```cpp
#include <iostream>
#include <string>
#include <fstream>

using namespace std;

const int MAX_VECTOR_SIZE = 100000000;
const int MAX_MATRIX_SIZE = 10000;

// Шаблон вектора
template <class ValType>
class TVector
{
protected:
    ValType* pVector;
    int Size;           // размер вектора
    int StartIndex;     // индекс первого элемента вектора
public:
    TVector(int s = 10, int si = 0);
    TVector(const TVector& v);                  // конструктор копирования
    ~TVector();                                 // Диструктор
    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
    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

    // векторные операции
    TVector  operator+(const TVector& v);       // сложение
    TVector  operator-(const TVector& v);       // вычитание
    ValType  operator*(const TVector& v);       // скалярное произведение

    // ввод-вывод
    friend istream& operator>>(istream& in, TVector& v) { ... }
    friend ostream& operator<<(ostream& out, const TVector& v) { ... }
};

template <class ValType>
TVector<ValType>::TVector(int s, int si)
{
    if (s > MAX_VECTOR_SIZE || s < 0)
    {
        bad_alloc exp;
        throw exp;
    }
    if ((si < 0) || (si > MAX_VECTOR_SIZE))
    {
        throw "Start index is negative";
    }
    pVector = new ValType[s];
    Size = s;
    StartIndex = si;
    for (int i = 0; i < s; i++)
        pVector[i] = 0;
}

template <class ValType> // Конструктор копирования
TVector<ValType>::TVector(const TVector<ValType>& v)
{
    Size = v.Size;
    StartIndex = v.StartIndex;
    pVector = new ValType[Size];
    copy(v.pVector, v.pVector + Size, pVector);
}

template <class ValType>
TVector<ValType>::~TVector()
{
    delete[] pVector;
}

template <class ValType> // Доступ
ValType& TVector<ValType>::operator[](int pos)
{
    if (pos < 0 || pos - StartIndex > Size)
    {
        throw "index out of range";
    }
    else
        return pVector[pos - StartIndex];
}
```

```cpp
template <class ValType> // Сравнение
bool TVector<ValType>::operator==(const TVector& v) const
{
    if (Size != v.Size)
        return false;
    for (int i = 0; i < Size; i++)
        if (pVector[i] != v.pVector[i])
            return false;
    return true;


}

template <class ValType> // Сравнение
bool TVector<ValType>::operator!=(const TVector& v) const
{
    return !(operator==(v));
}

template <class ValType> // Присваивание
TVector<ValType>& TVector<ValType>::operator=(const TVector& v)
{
    if (this != &v)
    {
        if (Size != v.Size)
        {
            Size = v.Size;
            delete[] pVector;
            pVector = new ValType[Size];
        }
        StartIndex = v.StartIndex;
        copy(v.pVector, v.pVector + Size, pVector);
    }
    return *this;
}

template <class ValType> // Прибавить скаляр
TVector<ValType> TVector<ValType>::operator+(const ValType& val)
{
    TVector<ValType> temp(*this);
    for (int i = 0; i < Size; i++)
        temp.pVector[i] += val;
    return temp;
}

template <class ValType> // Вычесть скаляр
TVector<ValType> TVector<ValType>::operator-(const ValType& val)
{
    TVector<ValType> temp(*this);
    for (int i = 0; i < Size; i++)
        temp.pVector[i] -= val;
    return temp;
}

template <class ValType> // Умножить на скаляр
TVector<ValType> TVector<ValType>::operator*(const ValType& val)
{
    TVector<ValType> temp(*this);
    for (int i = 0; i < Size; i++)
        temp.pVector[i] *= val;
    return temp;
}

template <class ValType> // Сложение
TVector<ValType> TVector<ValType>::operator+(const TVector<ValType>& v)
{
    if (Size == v.Size)
    {
        TVector<ValType> temp(Size, StartIndex);
        for (int i = 0; i < Size; i++)
            temp.pVector[i] = pVector[i] + v.pVector[i];
        return temp;
    }
    else
    {
        throw "not equal size!";
    }
}

template <class ValType> // Вычитание
TVector<ValType> TVector<ValType>::operator-(const TVector<ValType>& v)
{
    if (Size == v.Size)
    {
        TVector<ValType> temp(Size, StartIndex);
        for (int i = 0; i < Size; i++)
            temp.pVector[i] = pVector[i] - v.pVector[i];
        return temp;
    }
    throw "not equal size!";
}
```

```cpp
// Верхнетреугольная матрица
template <class ValType>
class TMatrix : public TVector<TVector<ValType> >
{
public:
    TMatrix(int s = 10);
    TMatrix(const TMatrix& mt);                    // копирование
    TMatrix(const TVector<TVector<ValType> >& mt); // преобразование типа
    bool operator==(const TMatrix& mt) const;      // сравнение
    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);        // перемножить

    TMatrix  operator+ (const ValType& num);
    TMatrix  operator- (const ValType& num);
    TMatrix  operator* (const ValType& num);

    // Ввод/Вывод
    friend istream& operator>>(istream& in, TMatrix& mt) { ... }
    friend ostream& operator<<(ostream& out, const TMatrix& mt)
    {
        for (int i = 0; i < mt.Size; i++)
            out << mt.pVector[i] << endl;
        return out;
    }

    void WriteFile(std::string path) { ... }

    void ReadFile(std::string path, TMatrix& mt)    // Чтение с файла
    {
        std::ifstream file(path, std::ios::in);
        if (file.is_open())
        {
            std::string buf;

            int val[250];
            int i_val = 0;
            std::string str;

            while (std::getline(file, buf))
            {
                for (int i = 0; i < buf.size(); i++)
                {
                    if (buf[i] == ' ')
                    {
                        val[i_val] = std::atoi(str.c_str());
                        i_val++;
                        str.clear();
                        continue;
                    }
                    str += buf[i];
                }
            }
            i_val = 0;
            for (int i = 0; i < 5; i++)
                for (int j = i; j < 5; j++)
                {
                    mt.pVector[i][j] = val[i_val];
                    i_val++;
                }
        }
    }
};

template <class ValType>
TMatrix<ValType>::TMatrix(int s) : TVector<TVector<ValType> >(s)
{
    if (s > MAX_MATRIX_SIZE || s < 0)
    {
        bad_alloc exp;
        throw exp;
    }
    for (int i = 0; i < s; i++)
    {
        this->pVector[i] = TVector<ValType>(s - i, i);
    }
}
```

```cpp
template <class ValType> // Конструктор копирования
TMatrix<ValType>::TMatrix(const TMatrix<ValType>& mt) :
    TVector<TVector<ValType> >(mt) {}

template <class ValType> // Конструктор преобразования типа
TMatrix<ValType>::TMatrix(const TVector<TVector<ValType> >& mt) :
    TVector<TVector<ValType> >(mt) {}

template <class ValType> // Сравнение
bool TMatrix<ValType>::operator==(const TMatrix<ValType>& mt) const
{
    return TVector<TVector<ValType>>::operator==(mt);
} /*-------------------------------------------------------------------------*/

template <class ValType> // Сравнение
bool TMatrix<ValType>::operator!=(const TMatrix<ValType>& mt) const
{
    return !(operator==(mt));
} /*-------------------------------------------------------------------------*/

template <class ValType> // Присваивание
TMatrix<ValType>& TMatrix<ValType>::operator=(const TMatrix<ValType>& mt)
{
    TVector < TVector<ValType>>::operator=(mt);
    return *this;
} /*-------------------------------------------------------------------------*/

template <class ValType> // Сложение
TMatrix<ValType> TMatrix<ValType>::operator+(const TMatrix<ValType>& mt)
{
    return TVector<TVector<ValType>>::operator+(mt);
} /*-------------------------------------------------------------------------*/

template <class ValType> // Вычитание
TMatrix<ValType> TMatrix<ValType>::operator-(const TMatrix<ValType>& mt)
{
    return TVector<TVector<ValType>>::operator-(mt);
} /*-------------------------------------------------------------------------*/

template<class ValType>
inline TMatrix<ValType> TMatrix<ValType>::operator*(const TMatrix& mt)
{
    if (this->Size != mt.Size)
        throw "Error size";
    TMatrix <ValType> resault(this->Size);
    for (int i = 0; i < this->Size; i++)
    {
        for (int j = i; j < this->Size; j++)
        {
            for (int k = i; k < j + 1; k++)
            {
                resault.pVector[i][j] += this->pVector[i][k] * mt.pVector[k][j];
            }
        }
    }
    return resault;
}

template<class ValType>
inline TMatrix<ValType> TMatrix<ValType>::operator*(const ValType& num)
{
    TMatrix res(this->Size);
    for (int i = 0; i < this->Size; i++)
        res.pVector[i] = TMatrix::pVector[i] * num;
    return res;
}

template<class ValType>
inline TMatrix<ValType> TMatrix<ValType>::operator+(const ValType& num)
{
    TMatrix res(this->Size);
    for (int i = 0; i < this->Size; i++)
        res.pVector[i] = TMatrix::pVector[i] + num;
    return res;
}

template<class ValType>
inline TMatrix<ValType> TMatrix<ValType>::operator-(const ValType& num)
{
    TMatrix res(this->Size);
    for (int i = 0; i < this->Size; i++)
        res.pVector[i] = TMatrix::pVector[i] - num;
    return res;
}
```

```cpp
// Нижни треугольные
template <class ValType>
class LowerMatrix : public TVector<TVector<ValType> >
{
public:
    LowerMatrix(int s = 10);
    LowerMatrix(const LowerMatrix& mt);                    // копирование
    LowerMatrix(const TVector<TVector<ValType> >& mt); // преобразование типа
    bool operator==(const LowerMatrix& mt) const;      // сравнение
    bool operator!=(const LowerMatrix& mt) const;      // сравнение
    LowerMatrix& operator= (const LowerMatrix& mt);        // присваивание
    LowerMatrix  operator+ (const LowerMatrix& mt);        // сложение
    LowerMatrix  operator- (const LowerMatrix& mt);        // вычитание
    LowerMatrix  operator* (const LowerMatrix& mt);        // перемножить

    LowerMatrix  operator+ (const ValType& num);
    LowerMatrix  operator- (const ValType& num);
    LowerMatrix  operator* (const ValType& num);

    // Ввод/Вывод
    friend istream& operator>>(istream& in, LowerMatrix& mt)
    {
        for (int i = 0; i < mt.Size; i++)
            in >> mt.pVector[i];
        return in;
    }
    friend ostream& operator<<(ostream& out, const LowerMatrix& mt)
    {
        for (int i = 0; i < mt.Size; i++)
            out << mt.pVector[i] << endl;
        return out;
    }

    void WriteFile(std::string path) { ... }

    void ReadFile(std::string path, LowerMatrix& mt) { ... }
};

template <class ValType>
LowerMatrix<ValType>::LowerMatrix(int s) : TVector<TVector<ValType> >(s)
{
    if (s > MAX_MATRIX_SIZE || s < 0)
    {
        bad_alloc exp;
        throw exp;
    }
    for (int i = 0; i < s; i++)
    {
        this->pVector[i] = TVector<ValType>(i + 1, 0);
    }
}

template <class ValType> // Конструктор копирования
LowerMatrix<ValType>::LowerMatrix(const LowerMatrix<ValType>& mt) :
    TVector<TVector<ValType> >(mt) {}

template <class ValType> // Конструктор преобразования типа
LowerMatrix<ValType>::LowerMatrix(const TVector<TVector<ValType> >& mt) :
    TVector<TVector<ValType> >(mt) {}

template <class ValType> // Сравнение
bool LowerMatrix<ValType>::operator==(const LowerMatrix<ValType>& mt) const
{
    return TVector<TVector<ValType>>::operator==(mt);
} /*-------------------------------------------------------------------------*/

template <class ValType> // Сравнение
bool LowerMatrix<ValType>::operator!=(const LowerMatrix<ValType>& mt) const
{
    return !(operator==(mt));
} /*-------------------------------------------------------------------------*/

template <class ValType> // Присваивание
LowerMatrix<ValType>& LowerMatrix<ValType>::operator=(const LowerMatrix<ValType>& mt)
{
    TVector < TVector<ValType>>::operator=(mt);
    return *this;
} /*-------------------------------------------------------------------------*/

template <class ValType> // Сложение
LowerMatrix<ValType> LowerMatrix<ValType>::operator+(const LowerMatrix<ValType>& mt)
{
    return TVector<TVector<ValType>>::operator+(mt);
} /*-------------------------------------------------------------------------*/

template <class ValType> // Вычитание
LowerMatrix<ValType> LowerMatrix<ValType>::operator-(const LowerMatrix<ValType>& mt)
{
    return TVector<TVector<ValType>>::operator-(mt);
} /*-------------------------------------------------------------------------*/
```

```cpp
template<class ValType>
inline LowerMatrix<ValType> LowerMatrix<ValType>::operator*(const LowerMatrix& mt)
{
    if (this->Size != mt.Size)
        throw "Error size";
    LowerMatrix <ValType> resault(this->Size);
    for (int i = 0; i < this->Size; i++)
    {
        for (int j = i; j < this->Size; j++)
        {
            for (int k = i; k < j + 1; k++)
            {
                resault.pVector[i][j] += this->pVector[i][k] * mt.pVector[k][j];
            }
        }
    }
    return resault;
}

template<class ValType>
inline LowerMatrix<ValType> LowerMatrix<ValType>::operator*(const ValType& num)
{
    LowerMatrix res(this->Size);
    for (int i = 0; i < this->Size; i++)
        res.pVector[i] = LowerMatrix::pVector[i] * num;
    return res;
}

template<class ValType>
inline LowerMatrix<ValType> LowerMatrix<ValType>::operator+(const ValType& num)
{
    LowerMatrix res(this->Size);
    for (int i = 0; i < this->Size; i++)
        res.pVector[i] = LowerMatrix::pVector[i] + num;
    return res;
}

template<class ValType>
inline LowerMatrix<ValType> LowerMatrix<ValType>::operator-(const ValType& num)
{
    LowerMatrix res(this->Size);
    for (int i = 0; i < this->Size; i++)
        res.pVector[i] = LowerMatrix::pVector[i] - num;
    return res;
}

// Разрежённая матрица
template <class ValType>
class RazrMatrix : public TVector<TVector<ValType> >
{
private:
    int sR;
public:
    RazrMatrix(int r, int s = 10);
    RazrMatrix(const RazrMatrix& mt);                    // копирование
    RazrMatrix(const TVector<TVector<ValType> >& mt); // преобразование типа
    bool operator==(const RazrMatrix& mt) const;      // сравнение
    bool operator!=(const RazrMatrix& mt) const;      // сравнение
    RazrMatrix& operator= (const RazrMatrix& mt);        // присваивание
    RazrMatrix  operator+ (const RazrMatrix& mt);        // сложение
    RazrMatrix  operator- (const RazrMatrix& mt);        // вычитание
    RazrMatrix  operator* (const RazrMatrix& mt);          // перемножить

    RazrMatrix  operator+ (const ValType& num);
    RazrMatrix  operator- (const ValType& num);
    RazrMatrix  operator* (const ValType& num);

    int get_Sr() const { return sR; }

    void input(TVector<ValType> vals, TVector<ValType> colums, TVector<ValType> rows)
    {
        for (int i = 0; i < this->Size; i++)
        {
            this->pVector[0][i] = vals[i];
            this->pVector[1][i] = colums[i];
        }
        this->pVector[2] = rows;
    }
    void print()
    {
        for (int row = 0; row < this->sR; ++row)
        {
            for (int i = this->pVector[2][row]; i < this->pVector[2][row + 1]; ++i)
            {
                cout << this->pVector[0][i] << " ";
            }
            cout << "\n";
        }
    }
```

```
    // Ввод/Вывод
    friend istream& operator>>(istream& in, RazrMatrix& mt)
    {
        for (int i = 0; i < 3; i++)
            in >> mt.pVector[i];
        return in;
    }
    friend ostream& operator<<(ostream& out, const RazrMatrix& mt)
    {
        for (int i = 0; i < 3; i++)
            out << mt.pVector[i] << endl;
        return out;
    }

    void WriteFile(std::string path) { ... }

    void ReadFile(std::string path, RazrMatrix& mt) { ... }
};

template <class ValType>
RazrMatrix<ValType>::RazrMatrix(int r, int s) : TVector<TVector<ValType> >(s)
{
    if (s > MAX_MATRIX_SIZE || s < 0)
    {
        bad_alloc exp;
        throw exp;
    }
    sR = r;
    this->pVector[0] = TVector<ValType>(s, 0);
    this->pVector[1] = TVector<ValType>(s, 0);
    this->pVector[2] = TVector<ValType>(r, 0);
}

template <class ValType> // Конструктор копирования
RazrMatrix<ValType>::RazrMatrix(const RazrMatrix<ValType>& mt) :
    TVector<TVector<ValType> >(mt) {}

template <class ValType> // Конструктор преобразования типа
RazrMatrix<ValType>::RazrMatrix(const TVector<TVector<ValType> >& mt) :
    TVector<TVector<ValType> >(mt) {}

template <class ValType> // Сравнение
bool RazrMatrix<ValType>::operator==(const RazrMatrix<ValType>& mt) const
{
    return TVector<TVector<ValType>>::operator==(mt);
} /*-------------------------------------------------------------------------*/

template <class ValType> // Сравнение
bool RazrMatrix<ValType>::operator!=(const RazrMatrix<ValType>& mt) const
{
    return !(operator==(mt));
} /*-------------------------------------------------------------------------*/

template <class ValType> // Присваивание
RazrMatrix<ValType>& RazrMatrix<ValType>::operator=(const RazrMatrix<ValType>& mt)
{
    TVector < TVector<ValType>>::operator=(mt);
    return *this;
} /*-------------------------------------------------------------------------*/

template <class ValType> // Сложение
RazrMatrix<ValType> RazrMatrix<ValType>::operator+(const RazrMatrix<ValType>& mt)
{
    RazrMatrix<ValType> res(this->sR, this->Size);
    res.pVector[0] = this->pVector[0] + mt.pVector[0];
    res.pVector[1] = this->pVector[1];
    res.pVector[2] = this->pVector[2];

    return res;
} /*-------------------------------------------------------------------------*/

template <class ValType> // Вычитание
RazrMatrix<ValType> RazrMatrix<ValType>::operator-(const RazrMatrix<ValType>& mt)
{
    RazrMatrix<ValType> res(this->sR, this->Size);
    res.pVector[0] = this->pVector[0] - mt.pVector[0];
    res.pVector[1] = this->pVector[1];
    res.pVector[2] = this->pVector[2];

    return res;
} /*-------------------------------------------------------------------------*/
```

```cpp
template<class ValType>
inline RazrMatrix<ValType> RazrMatrix<ValType>::operator*(const RazrMatrix& mt)
{
    if (this->Size != mt.Size)
        throw "Error size";
    RazrMatrix <ValType> resault(this->sR,this->Size);
    for (int i = 0; i < this->Size; i++)
    {
        for (int j = 0; j < this->Size; j++)
        {
            resault.pVector[0][i] += this->pVector[0][i] * mt.pVector[0][j];
        }
    }
    resault.pVector[1] = this->pVector[1];
    resault.pVector[2] = this->pVector[2];
    return resault;
}

template<class ValType>
inline RazrMatrix<ValType> RazrMatrix<ValType>::operator*(const ValType& num)
{
    RazrMatrix <ValType> resault(this->sR, this->Size);
    for (int i = 0; i < this->Size; i++)
    {
        resault.pVector[0][i] = this->pVector[0][i] * num;
    }
    resault.pVector[1] = this->pVector[1];
    resault.pVector[2] = this->pVector[2];
    return resault;
}

template<class ValType>
inline RazrMatrix<ValType> RazrMatrix<ValType>::operator+(const ValType& num)
{
    RazrMatrix <ValType> resault(this->sR, this->Size);
    for (int i = 0; i < this->Size; i++)
    {
        resault.pVector[0][i] = this->pVector[0][i] + num;
    }
    resault.pVector[1] = this->pVector[1];
    resault.pVector[2] = this->pVector[2];
    return resault;
}

template<class ValType>
inline RazrMatrix<ValType> RazrMatrix<ValType>::operator-(const ValType& num)
{
    RazrMatrix <ValType> resault(this->sR, this->Size);
    for (int i = 0; i < this->Size; i++)
    {
        resault.pVector[0][i] = this->pVector[0][i] - num;
    }
    resault.pVector[1] = this->pVector[1];
    resault.pVector[2] = this->pVector[2];
    return resault;
}

//Tape Matrix
template <class ValType>
class TapeMatrix : public TVector<TVector<ValType> >
{
private:
    int sizeTape;   // Размер ленты
public:
    TapeMatrix(int sT_, int s = 10);
    TapeMatrix(const TapeMatrix& mt);                    // копирование
    TapeMatrix(const TVector<TVector<ValType> >& mt); // преобразование типа
    bool operator==(const TapeMatrix& mt) const;      // сравнение
    bool operator!=(const TapeMatrix& mt) const;      // сравнение
    TapeMatrix& operator= (const TapeMatrix& mt);        // присваивание
    TapeMatrix  operator+ (const TapeMatrix& mt);        // сложение
    TapeMatrix  operator- (const TapeMatrix& mt);        // вычитание
    TapeMatrix  operator* (const TapeMatrix& mt);          // перемножить

    TapeMatrix  operator+ (const ValType& num);
    TapeMatrix  operator- (const ValType& num);
    TapeMatrix  operator* (const ValType& num);

    // Ввод/Вывод
    friend istream& operator>>(istream& in, TapeMatrix& mt)
    {
        for (int i = 0; i < mt.Size; i++)
            in >> mt.pVector[i];
        return in;
    }
    friend ostream& operator<<(ostream& out, const TapeMatrix& mt)
    {
        for (int i = 0; i < mt.Size; i++)
            out << mt.pVector[i] << endl;
        return out;
    }

    void WriteFile(std::string path) { ... }

    void ReadFile(std::string path, TapeMatrix& mt) { ... }
};
```

```cpp
template <class ValType>
TapeMatrix<ValType>::TapeMatrix(int sT_, int s) : TVector<TVector<ValType> >(s)
{
    if (s > MAX_MATRIX_SIZE || s < 0)
    {
        bad_alloc exp;
        throw exp;
    }
    for (int i = 0; i < s; i++)
    {
        if (i - 1 < 0)
        {
            if (i + 1 < s)
            {
                this->pVector[i] = TVector<ValType>(sT_ - 1, i);
            }
            else
            {
                this->pVector[i] = TVector<ValType>(sT_ - 2, i);
            }
        }
        else
        {
            if (i + 1 < s)
            {
                this->pVector[i] = TVector<ValType>(sT_, i - 1);
            }
            else
            {
                this->pVector[i] = TVector<ValType>(sT_ - 1, i - 1);
            }

            //this->pVector[i] = TVector<ValType>(s, 0);
        }
        sizeTape = sT_;
    }
}

template <class ValType> // Конструктор копирования
TapeMatrix<ValType>::TapeMatrix(const TapeMatrix<ValType>& mt) :
    TVector<TVector<ValType> >(mt) {}

template <class ValType> // Конструктор преобразования типа
TapeMatrix<ValType>::TapeMatrix(const TVector<TVector<ValType> >& mt) :
    TVector<TVector<ValType> >(mt) {}

template <class ValType> // Сравнение
bool TapeMatrix<ValType>::operator==(const TapeMatrix<ValType>& mt) const
{
    return TVector<TVector<ValType>>::operator==(mt);
} /*-------------------------------------------------------------------------*/

template <class ValType> // Сравнение
bool TapeMatrix<ValType>::operator!=(const TapeMatrix<ValType>& mt) const
{
    return !(operator==(mt));
} /*-------------------------------------------------------------------------*/

template <class ValType> // Присваивание
TapeMatrix<ValType>& TapeMatrix<ValType>::operator=(const TapeMatrix<ValType>& mt)
{
    TVector < TVector<ValType>>::operator=(mt);
    return *this;
} /*-------------------------------------------------------------------------*/

template <class ValType> // Сложение
TapeMatrix<ValType> TapeMatrix<ValType>::operator+(const TapeMatrix<ValType>& mt)
{
    return TVector<TVector<ValType>>::operator+(mt);
} /*-------------------------------------------------------------------------*/

template <class ValType> // Вычитание
TapeMatrix<ValType> TapeMatrix<ValType>::operator-(const TapeMatrix<ValType>& mt)
{
    return TVector<TVector<ValType>>::operator-(mt);
} /*-------------------------------------------------------------------------*/

template<class ValType>
inline TapeMatrix<ValType> TapeMatrix<ValType>::operator*(const TapeMatrix& mt)
{
    if (this->Size != mt.Size)
        throw "Error size";
    TapeMatrix <ValType> resault(this->sizeTape, this->Size);
    for (int i = 0; i < this->Size; i++)
    {
        for (int j = i; j < this->Size; j++)
        {
            for (int k = i; k < j + 1; k++)
            {
                resault.pVector[i][j] += this->pVector[i][k] * mt.pVector[k][j];
            }
        }
    }
    return resault;
}
```

```cpp
template<class ValType>
inline TapeMatrix<ValType> TapeMatrix<ValType>::operator*(const ValType& num)
{
    TapeMatrix res(this->sizeTape, this->Size);
    for (int i = 0; i < this->Size; i++)
        res.pVector[i] = TapeMatrix::pVector[i] * num;
    return res;
}

template<class ValType>
inline TapeMatrix<ValType> TapeMatrix<ValType>::operator+(const ValType& num)
{
    TapeMatrix res(this->sizeTape, this->Size);
    for (int i = 0; i < this->Size; i++)
        res.pVector[i] = TapeMatrix::pVector[i] + num;
    return res;
}

template<class ValType>
inline TapeMatrix<ValType> TapeMatrix<ValType>::operator-(const ValType& num)
{
    TapeMatrix res(this->sizeTape, this->Size);
    for (int i = 0; i < this->Size; i++)
        res.pVector[i] = TapeMatrix::pVector[i] - num;
    return res;
}
```

Приложение 2

Main.cpp

```cpp
#include <iostream>
#include "UMatrix.h"

#define PATH "C:\\out.txt"

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

    setlocale(LC_ALL, "Russian");
    cout << "Тестирование программ поддержки представления треугольн
        << endl;
    for (i = 0; i < 5; i++)
        for (j = i; j < 5; j++)
        {
            a[i][j] = i * 10 + j;
            b[i][j] = (i * 10 + j) * 100;
        }
    c = a + b;
    cout << "Matrix a = " << endl << a << endl;
    cout << "Matrix b = " << endl << b << endl;
    cout << "Matrix c = a + b" << endl << c << endl;

    c = a * b;
    cout << "Matrix c = a * b" << endl << c << endl;

    c = a * 5;
    cout << "Matrix c = a * 5" << endl << c << endl;

    c = a + 5;
    cout << "Matrix c = a + 5" << endl << c << endl;

    c = a - 5;
    cout << "Matrix c = a - 5" << endl << c << endl;


    LowerMatrix<int> test(5);
    for (i = 0; i < 5; i++)
        for (j = 0; j <= i; j++)
        {
            test[i][j] = i * 2 + j + 1;
        }
    cout << test << endl;

    int size = 25;
    TVector<int> val(size);                 // Znacheniya
    TVector<int> col(size);                 // Index colums
    TVector<int> row(6);                    // Index row
    TVector<int> tmp(size);                 // For prosmotr isxodnyx
    int tmp_i = 0;
    int k = 0;
    srand(time(NULL));
    for (i = 0; i < 5; i++)
    {
        for (j = 0; j < 5; j++)
        {
            int num = 0 + rand() % 3;       // randomim 0..2
            tmp[tmp_i] = num;
            tmp_i++;
            if (num > 0)                    // pishem only null
            {
                val[k] = num;
                col[k] = j;
                k++;
            }
            row[i + 1] = k;
        }
    }
    RazrMatrix<int> razr(6, k);
    cout << "Inpud data: " << tmp << "\n";      // check input data
    razr.input(val, col, row);                  // input matrix

    cout << "\nRazryazgennie matrix: \n";
    cout << razr << endl;

    cout << "Klassic vid matrix: \n";
    razr.print();

    cout << "Razryazgennie matrix r = a + a: " << endl;
    RazrMatrix<int> r(6, k);
    r = razr + razr;
    r.print();

    cout << "Razryazgennie matrix r = a * a: " << endl;
    r = razr * razr;
    r.print();

    cout << "Razryazgennie matrix r = a * 5: " << endl;
    r = razr * 5;
    r.print();


    TapeMatrix<int> tape(3, 6);
    for (i = 0; i < 6; i++)
    {
        // randomim 0..5
        tape[i][i] = 0 + rand() % 5;
        if(i - 1 > 0)
        {
            tape[i][i - 1] = 0 + rand() % 5;
        }
        if(i + 1 < 6)
        {
            tape[i][i + 1] = 0 + rand() % 5;
        }
    }
    cout << "Tape Matrix: \n";
    cout << tape;

    TapeMatrix<int> t(3, 6);

    cout << "Tape matrix t = tape + tape: " << endl;
    t = tape + tape;
    cout << t;

    cout << "Tape matrix t = tape + 5: " << endl;
    t = tape + 5;
    cout << t;

    a.WriteFile(PATH);
    TMatrix<int> m(5);
    m.ReadFile(PATH, m);

    cout << "After read file m = " << endl << m << endl;

    system("pause");
    return 0;
}
```