Национальный исследовательский Нижегородский государственный университет имени Н. И. Лобачевского

Отчёт по лабораторной работе

Структуры хранения и операции над векторами и матрицами

Выполнил:  
студент ф-та ИИТММ, группа 3822Б1ФИ5  
Будыкин Рудольф-Евгений Александрович.

Проверил:  
Лебедев Илья Геннадьевич

Содержание:

1. [Введение](#Введение)
2. [Постановка задачи](#Постановка_задачи)
3. [Руководство пользователя](#Руководство_польз)
4. [Руководство программиста](#Руководство_прог)
5. [Описание структур данных](#Описание_структур)
6. [Описание алгоритмов](#Описание_алгоритмов)
7. [Тесты](#Тесты)
8. [Заключение](#Заключение)
9. [Литература](#Литература)
10. [Приложение](#Приложение)

**1. Введение**

Операции над векторами и матрицами являются весьма тривиальной задачей, но при большом количестве вычислений требуется облегчение процесса, ввиду невозможности держать в голове одновременно результаты всех действий даже над одним и тем же набором элементов.

**2. Постановка задачи**

Задачей является реализация программы (библиотеки) по работе с матрицами и векторами на основе классов. Обработка должна включать в себя операции с матрицами: умножение, сложение, вычитание, сравнение, аналогичные операции с векторами.

(Текущая версия программы (0.3) не имеет полного функционала, что указано в каждом из необходимых пунктов)

**3. Руководство пользователя (полная версия на релиз)**

Программа может выполнять определенные действия с векторами и матрицами, а именно:

• Создание и инициализация векторов и матриц разной размерности.

• Выполнение операций сложения, вычитания и умножения для векторов и матриц.

• Вывод результатов операций на экран.

**3.1 Руководство пользователя (версия 0.3)**

На данный момент реализованы следующие операции

* Сложение векторов
* Вычитание векторов
* Ввод/вывод результатов на экран отдельными функциями.

**4. Руководство программиста**

Программисту предоставляются библиотеки (файлы .h) включающие:

• TVector.h - Реализация класса для векторов. Операции над векторами.

• TMatrix.h - Класс для матриц, наследуемый от класса векторов. (в версии 0.3 не имеет поставленного задачей функционала)

• Тестовое приложение. После первого открытия кода в компиляторе необходимо выставить test\_MatrixLab в качестве запускаемого проекта. Приложение содержит ряд тестов демонстрирующих работу библиотек.

• Так же в проект включены source файлы TMatrix.cpp и TVector.cpp, необходимые для успешной сборки программы в cmake, функционала не несут ввиду полного включения оного в .h файлы.

**5. Описание структур данных**

1. Класс TVector и его элементы:

1. Поля класса:

* size – отвечает за размерность вектора.
* data – само содержание вектора (его числа по каждому из измерений).

1. Методы и конструкторы класса:

* Конструкторы по умолчанию, копирования, перемещения, перегрузки различных операторов для работы с векторами.
* Функции для непосредственной работы с данными вектора.

2. Класс TMatrix и его элементы

Для версии 0.3 реализован лишь как шаблон, не имеет смысла в представлении.

**6. Описание алгоритмов**

1. Оператор присваивания для TVector

• Алгоритм:

1. Проверка на самоприсваивание.
2. Проверка на непустое поле данных в присваиваемом векторе.
3. Обработка введенного вектора конвертером во избежание ошибок
4. Копирование элементов из исходного вектора в текущий.

2. Операторы сложения/вычитания для TVector

• Алгоритм:

1. Проверка на совпадение размерности оперируемых векторов
2. Непосредственные операции над векторами, соответствующие оператору.

3. Функция Vvod() – ввод вектора с клавиатуры упрощающий код main файла для удобства программиста.

4. Функция AddVec отвечает за ошибки определения памяти (если вектор создан с одной размерностью, и по причине какой то ошибки у него оказалась другая, функция исправит это прогнав через себя все данные вектора), необходима для избавления от мусора в данных после функции ввода

**7. Тесты**

Результаты экспериментов показали эффективность разработанных классов при выполнении операций с верхнетреугольными матрицами. Анализ выявил отсутствие нежелательных побочных эффектов и высокую стабильность работы программы.

**8. Заключение**

Лабораторная работа успешно реализовала программные средства для эффективного хранения и обработки верхнетреугольных матриц и векторов. В ходе тестирования была подтверждена корректность и работоспособность разработанных классов.

**9. 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.

**10.Приложение**

// Верхнетреугольная матрица - реализация на основе шаблона вектора

#ifndef \_\_TMATRIX\_H\_\_

#define \_\_TMATRIX\_H\_\_

#include <iostream>

#include <functional>

using namespace std;

const int MAX\_VECTOR\_SIZE = 100000000;

const int MAX\_MATRIX\_SIZE = 10000;

// Шаблон вектора

template <class T>

class TVector

{

protected:

T \*pVector;

int Size; // размер вектора

public:

typedef function<T(size\_t)> AllocatorFunc;

TVector(int s = 1, AllocatorFunc fnAlloc = nullptr);

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

~TVector();

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

T& operator[](int pos); // доступ

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

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

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

// скалярные операции

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

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

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

// векторные операции

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

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

T 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 T>

TVector<T>::TVector(int s = 1, AllocatorFunc fnAlloc = nullptr) : Size(s)

{

if (s < 0 || s > MAX\_VECTOR\_SIZE) { throw exception("Invalid size"); }

pVector = new T[Size]();

if (fnAlloc)

{

for (size\_t i = 0; i < Size; i++)

{

pVector[i] = fnAlloc(i);

}

}

} /\*-------------------------------------------------------------------------\*/

template <class T> //конструктор копирования

TVector<T>::TVector(const TVector<T> &v): Size(v.Size)

{

pVector = new T[Size]();

for (int i = 0; i < Size; i++) { pVector[i] = v.pVector[i]; }

} /\*-------------------------------------------------------------------------\*/

template <class T>

TVector<T>::~TVector()

{

delete[] pVector;

} /\*-------------------------------------------------------------------------\*/

template <class T> // доступ

T& TVector<T>::operator[](int pos)

{

if (pos < 0 || pos > Size) { throw exception("Invalid position"); }

return pVector[pos];

} /\*-------------------------------------------------------------------------\*/

template <class T> // сравнение

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

{

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 T> // сравнение

bool TVector<T>::operator!=(const TVector &v) const noexcept

{

return !(\*this == v);

} /\*-------------------------------------------------------------------------\*/

template <class T> // присваивание

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

{

if (this != &v)

{

if (Size != v.Size)

{

Size = v.Size;

delete[] pVector;

pVector = new T[Size];

}

copy(v.pVector, v.pVector + Size, pVector);

}

return \*this;

} /\*-------------------------------------------------------------------------\*/

template <class T> // прибавить скаляр

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

{

TVector<T> temp(Size);

for (int i = 0; i < Size; i++) { temp[i] = pVector[i] + val; }

return temp;

} /\*-------------------------------------------------------------------------\*/

template <class T> // вычесть скаляр

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

{

TVector<T> temp(Size);

for (int i = 0; i < Size; i++) { temp[i] = pVector[i] - val; }

return temp;

} /\*-------------------------------------------------------------------------\*/

template <class T> // умножить на скаляр

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

{

TVector<T> temp(Size);

for (int i = 0; i < Size; i++) { temp[i] = pVector[i] \* val; }

return temp;

} /\*-------------------------------------------------------------------------\*/

template <class T> // сложение

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

{

if (Size != v.Size) { throw exception("Invalid size"); }

TVector<T> temp(Size);

for (int i = 0; i < Size; i++) { temp.pVector[i] = pVector[i] + v.pVector[i]; }

return temp;

} /\*-------------------------------------------------------------------------\*/

template <class T> // вычитание

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

{

if (Size != v.Size) { throw exception("Invalid size"); }

TVector<T> temp(Size);

for (int i = 0; i < Size; i++) { temp.pVector[i] = pVector[i] - v.pVector[i]; }

return temp;

} /\*-------------------------------------------------------------------------\*/

template <class T> // скалярное произведение

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

{

if (Size != v.Size) { throw exception("Invalid size"); }

int result = 0;

for (int i = 0; i < Size; i++) { result += pVector[i] \* v.pVector[i]; }

return result;

} /\*-------------------------------------------------------------------------\*/

// Верхнетреугольная матрица

template <class T>

class TMatrix : public TVector<TVector<T>>

{

public:

TMatrix(int s = 5);

TMatrix(const TMatrix &mt); // копирование

TMatrix(const TVector<TVector<T> > &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);

// 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 T>

TMatrix<T>::TMatrix(int s) : TVector<TVector<T> >(s, [s](size\_t i) { return TVector<T>(s); })

{

if (s < 0 || s > MAX\_MATRIX\_SIZE) { throw exception("Invalid size"); }

} /\*-------------------------------------------------------------------------\*/

template <class T> // конструктор копирования

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

TVector<TVector<T> >(mt) {}

template <class T> // конструктор преобразования типа

TMatrix<T>::TMatrix(const TVector<TVector<T> > &mt):

TVector<TVector<T> >(mt) {}

template <class T> // сравнение

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

{

if (Size != mt.Size) { return false; }

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

{

if (pVector[i] != mt.pVector[i]) { return false; }

}

return true;

} /\*-------------------------------------------------------------------------\*/

template <class T> // сравнение

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

{

return !(\*this == mt);

} /\*-------------------------------------------------------------------------\*/

template <class T> // присваивание

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

{

if (this != &mt) { TVector<TVector<T>>::operator=(mt); }

return \*this;

} /\*-------------------------------------------------------------------------\*/

template <class T> // сложение

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

{

if (Size != mt.Size) { throw exception("Invalid size"); }

TMatrix<T> result(Size);

for (int i = 0; i < Size; i++) { result.pVector[i] = pVector[i] + mt.pVector[i]; }

return result;

} /\*-------------------------------------------------------------------------\*/

template <class T> // вычитание

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

{

if (Size != mt.Size) { throw exception("Invalid size"); }

TMatrix<T> result(Size);

for (int i = 0; i < Size; i++) { result.pVector[i] = pVector[i] - mt.pVector[i]; }

return result;

}

template<class T>

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

{

return TMatrix();

}

#endif

// Тестирование верхнетреугольной матрицы

#include <iostream>

#include "utmatrix.h"

//---------------------------------------------------------------------------

void main()

{

TMatrix<int> a(5), b(5), c(5);

int i, j;

setlocale(LC\_ALL, "Russian");

cout << "Тестирование программ поддержки представления треугольных матриц"

<< endl;

for (i = 0; i < 5; i++)

for (j = i; j < 5; j++ )

{

a[i][j] = i + j;

b[i][j] = i + j \* 10;

}

c = a + b;

cout << "Matrix a = " << endl << a << endl;

cout << "Matrix b = " << endl << b << endl;

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

}

//---------------------------------------------------------------------------