**Министерство науки и высшего образования Российской Федерации**

**Федеральное государственное автономное образовательное**

**учреждение высшего образования «Национальный исследовательский**

**Нижегородский государственный университет им. Н. И. Лобачевского»**

Институт информационных технологий, математики и механики

Прикладная математика и информатика

**Отчёт**

**Тема: шаблонные классы Вектор и Матрица**

Выполнил: студент группы 3822Б1ПМ3

Гуленков Леонид Сергеевич

Проверила: Шестакова Наталья Валерьевна

***Нижний Новгород, 2022***

**Содержание:**

1. Введение……………………………………………………………………………….. 3
2. Постановка задачи………………………………………………………………… 5
3. Руководство пользователя……………………………………………………. 6
4. Руководство программиста…………………………………………………… 10
5. Заключение……………………………………………………………………………. 15
6. Список литературы………………………………………………………………… 16
7. Приложение…………………………………………………………………………… 17

**Введение**

Матрица – это удобная и широко используемая структура данных в линейной алгебре, представляющая собой упорядоченный набор чисел, организованных в виде прямоугольной таблицы. Каждый элемент матрицы обозначается двумя индексами: номером строки и номером столбца.

В зависимости от свойств и структуры элементов матрицы, они могут классифицироваться по разным критериям:

* По количеству строк и столбцов:
  + Прямоугольные матрицы
  + Квадратные матрицы
* По свойствам элементов:
  + Диагональные матрицы (все элементы вне главной диагонали равны нулю)
  + Треугольные матрицы (все элементы выше или ниже главной диагонали равны нулю)
* По содержимому:
  + Нулевая матрица (все элементы матрицы равны нулю)
  + Единичная матрица: (главная диагональ состоит из единиц, остальные элементы – нули)
* По структуре:
  + Симметричные матрицы (элементы симметричны относительно главной диагонали)

Определение матрицы возможно также через понятие Вектор.

Вектор в математике – набор чисел (математических выражений), состоящий из n элементов. Тогда Матрица из m строк и n столбцов может быть определена как Вектор из m элементов, где каждый элемент, в свою очередь, является вектором из n элементов.

Базовые операции, которые можно выполнять над векторами:

* Сложение векторов: операция, при которой каждый элемент одного вектора суммируется с соответствующим элементом другого вектора, что приводит к получению нового вектора с тем же количеством элементов.
* Умножение вектора на число: операция, при которой каждый элемент вектора умножается на заданное число, что приводит к изменению масштаба вектора.
* Скалярное умножение векторов: операция, при которой умножаются соответствующие элементы двух векторов, а затем полученные произведения суммируются. Результатом является число, которое может использоваться, например, для вычисления угла между векторами.
* Сравнение векторов: вектора считаются равными тогда и только тогда, когда соответствующие компоненты векторов равны друг другу.
* Инкрементирование вектора: операция, при которой каждый элемент вектора увеличивается на заданную величину.

Базовые операции над матрицами:

* Сравнение матриц: проверка равенства двух матриц происходит покомпонентно. Две матрицы считаются равными, если соответствующие элементы обеих матриц равны.
* Сложение и вычитание матриц: происходит покомпонентно, то есть каждый элемент суммируется или вычитается с соответствующим элементом другой матрицы.
* Сложение и вычитание с числами: к каждому элементу прибавляется или вычитается число. Операция выполняется над каждым элементом матрицы.
* Умножение матрицы на число: каждый элемент матрицы умножается на заданное число.
* Умножение матрицы на матрицу: результат умножения матрицы на матрицу получается путем скалярного произведения строк первой матрицы на столбцы второй матрицы. Размерность результирующей матрицы определяется числом строк первой матрицы и числом столбцов второй.

Эти операции играют ключевую роль в линейной алгебре и широко применяются в различных областях, включая физику, статистику, компьютерную графику и многие другие.

**Постановка задачи**

В рамках лабораторной работы ставится задача создания программных средств, осуществляющих хранение квадратных матриц в виде вектора векторов и выполнение основных операций над ними:

* Сложение/вычитание
* Умножение
* Копирование
* Присваивание
* Сравнение
* Вывод на экран компонент матрицы в виде отдельных элементов или строк

Программные средства должны содержать:

* Шаблон класса Вектор
* Шаблон класса Матрица
* Тестовое приложение, позволяющее задавать матрицы и осуществлять основные операции над ними

Как уже было замечено ранее, работа с матрицами ограничивается использованием лишь квадратных матриц.

**Руководство пользователя**

Вектор

Созданный шаблонный класс векторов TVector позволяет проводить над ними операции. Для начала, необходимо создать как минимум один объект класса, используя один из конструкторов, не забыв указать в угловых скобках тип данных компонент вектора (например, int или double).

* Конструктор инициализации



В параметры можно передать как один аргумент (размерность вектора), так и два (размерность и массив, содержащий значения компонент).

* Конструктор по умолчанию



Ничего не передается в параметры. Размерность вектора по умолчанию становится равна единице, компоненты отсутствуют.

* Конструктор копирования



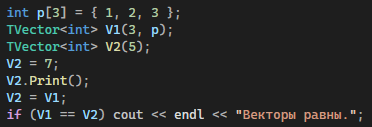
Теперь, когда в нашем распоряжении несколько экземпляров класса, мы можем совершать операции над ними.

Можно вывести как отдельную компоненту вектора на экран, так и весь вектор сразу. При этом, с помощью операции [ ] есть возможность изменять значения компонент вектора.

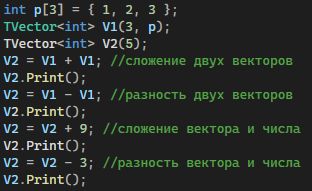
 

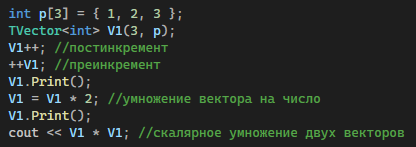
С помощью метода Input() можно ввести значения компонент вектора через консоль.

Метод GetSize() выводит на экран размер вектора.  

Одному вектору можно присвоить другой вектор или число. Также, можно проверять их на равенство.  

Доступны базовые арифметические операции над векторами. Применение и вывод результата продемонстрированы на скриншотах:

Матрица

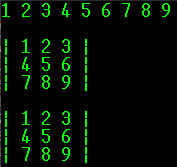
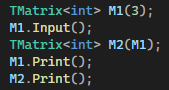
Для того, чтобы начать работать с матрицами, необходимо совершить аналогичные действия.

Созданный шаблонный класс матриц TMatrix позволяет проводить над ними операции. Для начала, необходимо создать как минимум один объект класса, используя один из конструкторов, не забыв указать в угловых скобках тип данных элементов матрицы (например, int или double):

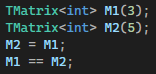
* Конструктор инициализации – в параметры нужно передать один аргумент – размерность матрицы. Изначально все элементы матрицы равны нулю.
* Конструктор копирования – в параметры нужно передать копируемую матрицу. Все элементы изначальной матрицы становятся равны соответствующим элементам копируемой матрицы.

После того как в нашем распоряжении оказалось несколько экземпляров класса, мы можем совершать операции над ними.

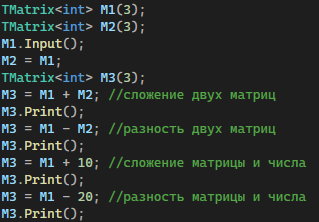
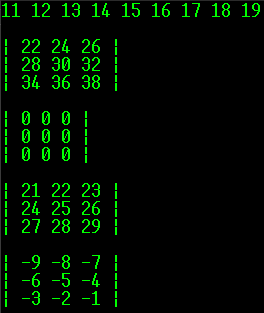
Чтобы заполнить только что созданную нулевую матрицу или обновить значения уже существующей используется метод Input(), который запрашивает ввод через консоль значение для каждого элемента.

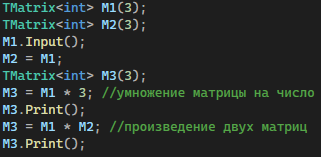
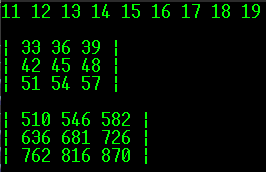
Чтобы вывести матрицу на экран используется метод Print().

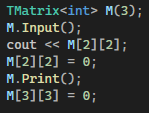
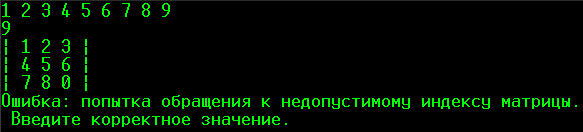
С помощью оператора = одной матрице можно присваивать значения всех элементов другой матрицы.

Также, можно сравнивать матрицы на равенство. Матрицы равны тогда и только тогда, когда равны их размерности и соответствующие элементы.  

Доступны базовые арифметические операции над матрицами. Применение и вывод результата продемонстрированы на скриншотах:

Можно вывести отдельный элемент матрицы на экран. При этом, с помощью операции [ ] есть возможность изменять значения элементов. При попытке обратиться к несуществующему элементу – программа будет ожидать повторный ввод.  

**Руководство программиста**

Описание структуры данных.

Структура данных — это способ организации и хранения данных, который обеспечивает эффективное выполнение определенных операций. У этих данных есть свой уникальный набор значений, отношений между ними и операций, которые могут быть применены к этим значениям.

* Шаблонный класс векторов **TVector**, имеющий два защищенных (**protected**) поля: **size** (определяет размерность вектора) и **Type\* val** (определяет значения компонент вектора), где **Type** – параметр шаблона. В качестве множества значений представлен упорядоченный набор элементов, содержащихся внутри вектора. В качестве операций заданы операции линейной алгебры над векторами. В качестве отношений задано отношение следования элементов вектора.
* Шаблонный класс матриц **TMatrix**. Класс использует приватное (**private**) наследование от класса **TVector<TVector<Type>>**. Это означает, что все его публичные (**public**) и защищенные (**protected**) члены, кроме конструкторов и деструкторов, становятся приватными членами **TMatrix**. В качестве множества значений представлен упорядоченный набор элементов, содержащихся внутри матрицы. В качестве операций заданы операции линейной алгебры над матрицами. В качестве отношений задано отношение следования элементов матрицы в строке и в столбце.
* using **TVector<TVector<Type>>::size** **и using TVector<TVector<Type>>::val** предоставляют доступ к членам **size** и **val** базового класса.

Описание алгоритмов.

В классе **TVector** доступны следующие методы и перегрузки:

* Конструктор по умолчанию. Переменная **size** приравнивается к единице.
* Конструктор инициализации. Переменная **size** задается пользователем. Для **Type\* val** выделяется соответствующая память с помощью **new Type[size]**. В случае, если пользователь также передает массив значений, он передаётся в **Type\* val**. Иначе, значения становятся равными нулю.
* Конструктор копирования. В качестве аргумента передается другой вектор. Значения **size** и **Type\* val** становятся равны значениям переданного вектора (с помощью цикла **for**).
* Деструктор. Вызывается автоматически, когда объект выходит за пределы области видимости. С помощью **delete[] val** освобождается память. Переменная **size** приравнивается к нулю.
* Метод Print() для вывода значений вектора на экран с помощью цикла **for** (от **0** до **size**).
* Перегрузка оператора **=**. В качестве аргумента передается вектор. Используется **delete[] val** для освобождения памяти выделенной для старых значений, размерность **size** становится равна размерности переданного вектора, а затем с помощью цикла **for** (от **0** до **size**) значения компонент вектора становятся равны значениям переданного.
* Перегрузка оператора +. В качестве аргумента передается вектор. С помощью конструктора инициализации создается новый вектор **result**. Если размерности векторов совпадают, то с помощью цикла **for** компоненты нового вектора становятся равны сумме соответствующих компонент двух складывающихся векторов. Возвращается вектор **result**. Иначе, на экран выводится сообщение о том, что векторы имеют разную размерность и возвращается нулевой вектор.
* Перегрузка постфиксной и префиксной инкрементации. С помощью цикла **for** (от **0** до **size**) значение каждой компоненты увеличивается на единицу. В случае префиксной инкрементации возвращается уже измененный вектор. В случае постфиксной инкрементации перед циклом **for** конструктором копирования создается новый вектор, равный старому до его изменения, который впоследствии и возвращается.
* Перегрузка оператора ==. В качестве аргумента передается вектор. Если размерности векторов совпадают, то с помощью цикла **for** проверяется равенство соответствующих координат векторов. В случае неравенства хотя бы одной пары соответствующих координат возвращается **false**. Если все значения попарно равны, возвращается **true**.
* Перегрузка оператора \*. В случае, если передается значение типа **double**, с помощью конструктора копирования создается новый вектор **res**, равный изначальному. Затем, с помощью цикла **for** каждая его компонента умножается на переданное значение. Возвращается вектор **res**. В случае, если передается другой вектор, выполняется скалярное умножение. Создается переменная **res**, равная нулю. Затем, если размерности векторов совпадают (иначе, на экран выводится сообщение о том, что скалярное произведение невозможно), с помощью цикла **for** к переменной **res** на каждой итерации прибавляется произведение соответствующих компонент векторов. Возвращается переменная **res**.
* Перегрузка оператора [ ]. В качестве аргумента передается целое число. Если это число больше или равно нулю и меньше **size**, то возвращается значение соответствующей координты вектора. Иначе, на экран выводится сообщение о попытке взять значение по несуществующему индексу и программа запрашивает ввод до тех пор, пока не будет передан существующий индекс.

В классе **TMatrix** доступны следующие методы и перегрузки:

* Конструктор инициализации. Инициализирует матрицу заданного размера. Сначала вызывается конструктор базового класса **TVector<TVector<Type>>(size)**, устанавливая размерность внешнего вектора (количество строк). Затем с использованием цикла **for** создаются внутренние вектора типа **TVector<Type>** с тем же размером, формируя столбцы матрицы. В результате получается квадратная матрица, представленная вектором векторов.
* Конструктор копирования. Принимает объект **mat** в качестве аргумента и вызывает конструктор копирования базового класса **TVector<TVector<Type>>(mat)**. Это позволяет скопировать размерность и значения вектора из объекта **mat** в текущий объект **TMatrix**.
* Перегрузка оператора ==. Если размерности матриц совпадают, то используется цикл **for** для итерации по элементам векторов (векторам) и сравнения их соответствующих компонент (перегрузка оператора == класса **TVector**). По завершении сравнения, выводится сообщение в консоль о том, равны матрицы или нет. Затем возвращается логическое значение **answer**.
* Перегрузка оператора =. Копирует значения из матрицы **mat** в текущий объект **TMatrix**. Он начинает с установки размерности матрицы (**size**) равной размерности матрицы **mat**. Затем с использованием цикла копируются значения векторов (перегрузка оператора = класса **TVector**) из **mat** в текущий объект. В конце метод возвращает указатель на текущий объект **\*this**.
* Перегрузка оператора + и – (параметром является матрица). Создает новую матрицу **Res**, которая будет результатом сложения или вычитания текущей матрицы и матрицы **mat**. Перед созданием **Res** проверяется, что размерности матриц совпадают. Если размерности совпадают, то с использованием цикла **for** происходит поэлементное сложение или вычитание векторов (перегрузка оператора + или - класса **TVector**) соответствующих компонент матриц (векторов), и результат сохраняется в матрицу **Res**. Если размерности не совпадают, выводится сообщение об ошибке, и возвращается матрица **Res**.
* Перегрузка оператора + и – (параметром является число). Выполняется поэлементное сложение или вычитание (перегрузка оператора + или - класса **TVector**) текущей матрицы с числом **num**. Создается новая матрица **Res**, и с использованием цикла **for** к каждому элементу (вектору) текущей матрицы прибавляется значение **num**. Результат сохраняется в соответствующий элемент матрицы **Res**. Затем созданная матрица **Res** возвращается из метода.
* Перегрузка оператора \* (параметром является число). Выполняется поэлементное умножение текущей матрицы на число **num**. Создается новая матрица **Res**, и с использованием цикла **for** каждый элемент (вектор) текущей матрицы умножается (перегрузка оператора \* класса **TVector**) на значение **num**. Результат сохраняется в соответствующий элемент матрицы **Res**. Затем созданная матрица **Res** возвращается из метода.
* Перегрузка оператора \* (параметром является матрица). Выполняется умножение текущей матрицы на матрицу **mat**. Создается новая матрица **result** размерности **size**. Если размерности матриц несовместимы для умножения, выводится сообщение об ошибке, и возвращается нулевая матрица. В противном случае, с использованием вложенных циклов **for** выполняется стандартный алгоритм умножения матриц. Каждый элемент **result** вычисляется как сумма произведений соответствующих элементов текущей матрицы и матрицы **mat**. В конце метода созданная матрица **result** возвращается.
* Перегрузка оператора [ ]. Возвращается ссылка на вектор по указанному индексу **index**. Перед возвратом выполняется проверка, что **index** находится в допустимых границах (от **0** до **size-1**). Если индекс выходит за пределы допустимого диапазона, выводится сообщение об ошибке, и пользователю предлагается ввести новое значение **index**.

Описание структуры программы.

* TVector.h, TVector.cpp – шаблонный класс, реализующий структуру данных Вектор.
* TMatrix.h, TMatrix.cpp – шаблонный класс, реализующий структуру данных Матрица.
* Source.cpp – исполнительный файл, запускающий программу тестирования.

**Заключение**

Мною реализованы шаблонные классы Вектор и Матрица. В программе демонстрируется работоспособность методов классов, а также базовых операций над векторами и матрицами для различных типов данных (системных и пользовательских).

**Список литературы**

https://habr.com/ru/articles/599801/

**Приложение**

**TVector.h**

#pragma once

#include <iostream>

using namespace std;

template <typename Type>

class TVector {

protected:

int size;

Type\* val;

public:

explicit TVector(int n) {

size = n;

val = new Type[size];

//if (size > 0) cout << "Значения компонент по умолчанию равны нулю." << endl;

for (int i = 0; i < size; i++) {

val[i] = 0;

}

};

TVector(int n, Type\* p) {

size = n;

val = new Type[size];

for (int i = 0; i < size; i++) {

val[i] = p[i];

}

};

TVector(const TVector<Type>& vec) {

size = vec.size;

val = new Type[size];

for (int i = 0; i < size; i++) {

val[i] = vec.val[i];

}

};

~TVector() {

delete[] val;

size = 0;

};

TVector() {

size = 1;

val = new Type[size];

};

void Print() {

cout << "| ";

for (int i = 0; i < size; i++) {

cout << val[i] << " ";

}

cout << "|" << endl;

};

Type& operator[](int index) {

while (index >= size || index < 0) {

cout << "Ошибка: попытка взять значение по несуществующему индексу. Введите корректное значение." << endl;

cin >> index;

}

return val[index];

};

Type& operator[](int index) const {

while (index >= size || index < 0) {

cout << "Ошибка: попытка взять значение по несуществующему индексу. Введите корректное значение." << endl;

cin >> index;

}

return val[index];

};

TVector& operator=(const TVector& vec) {

delete[] val;

size = vec.size;

val = new Type[size];

for (int i = 0; i < size; i++) {

val[i] = vec.val[i];

}

return \*this;

};

TVector& operator=(Type num) {

for (int i = 0; i < size; i++) {

val[i] = num;

}

return \*this;

};

TVector operator+(const TVector& vec) {

TVector result(vec.size);

if (size == vec.size) {

for (int i = 0; i < size; i++) {

result.val[i] = val[i] + vec.val[i];

}

}

else cout << "Векторы имеют разную размерность. Сложение невозможно." << endl << "Возвращён нулевой вектор." << endl;

return result;

};

TVector operator-(const TVector& vec) {

TVector result(vec.size);

if (size == vec.size) {

for (int i = 0; i < size; i++) {

result.val[i] = val[i] - vec.val[i];

}

}

else cout << "Векторы имеют разную размерность. Вычитание невозможно." << endl << "Возвращён нулевой вектор." << endl;

return result;

};

TVector operator+(Type num) {

TVector result(size);

for (int i = 0; i < size; i++) {

result.val[i] = val[i] + num;

}

return result;

};

TVector operator-(Type num) {

TVector result(size);

for (int i = 0; i < size; i++) {

result.val[i] = val[i] - num;

}

return result;

};

TVector& operator++() {

for (int i = 0; i < size; i++) {

val[i] = val[i] + 1;

}

return \*this;

};

TVector operator++(int) {

TVector res(\*this);

for (int i = 0; i < size; i++) {

val[i] = val[i] + 1;

}

return res;

};

TVector operator\*(double num) {

TVector res(\*this);

for (int i = 0; i < size; i++) {

res.val[i] \*= num;

}

return res;

};

double operator\*(const TVector& vec) {

double res = 0;

if (size == vec.size) {

for (int i = 0; i < size; i++) {

res += val[i] \* vec.val[i];

}

}

else cout << "Векторы имеют разную размерность. Скалярное" << endl << "произведение невозможно. Возвращён ноль." << endl;

return res;

};

bool operator==(const TVector& vec) {

if (size == vec.size) {

for (int i = 0; i < size; i++) {

if (val[i] != vec.val[i]) return false;

}

}

else {

return false;

}

return true;

};

int GetSize() {

return size;

};

void Input() {

for (int i = 0; i < size; i++) {

cin >> val[i];

}

};

};

**TMatrix.h**

#pragma once

#include <iostream>

#include <locale.h>

#include "TVector.h"

using namespace std;

template<typename Type>

class TMatrix : private TVector<TVector<Type>> {

using TVector<TVector<Type>>::size;

using TVector<TVector<Type>>::val;

public:

TMatrix(int size = 1) : TVector<TVector<Type>>(size) {

for (int i = 0; i < size; i++) {

val[i] = TVector<Type>(size);

}

};

TMatrix(const TMatrix& mat) :TVector<TVector<Type>>(mat) {};

bool operator==(const TMatrix& mat) {

bool answer = true;

if (size == mat.size) {

for (int i = 0; i < size; i++) {

if (!(val[i] == mat.val[i])) {

answer = 0;

break;

}

}

}

else answer = false;

if (answer) cout << "Матрицы равны." << endl;

else cout << "Матрицы НЕ равны." << endl;

return answer;

};

TMatrix& operator=(const TMatrix& mat) {

size = mat.size;

for (int i = 0; i < size; i++) {

val[i] = mat.val[i];

}

return \*this;

};

TMatrix operator+(const TMatrix& mat) {

TMatrix Res(size);

if (size == mat.size) {

for (int i = 0; i < size; i++) {

Res.val[i] = val[i] + mat.val[i];

}

}

else cout << "Матрицы имеют разную размерность. Сложение невозможно." << endl;

return Res;

};

TMatrix operator-(const TMatrix& mat) {

TMatrix Res(size);

if (size == mat.size) {

for (int i = 0; i < size; i++) {

Res.val[i] = val[i] - mat.val[i];

}

}

else cout << "Матрицы имеют разную размерность. Вычитание невозможно." << endl;

return Res;

};

TMatrix operator+(Type num) {

TMatrix Res(size);

for (int i = 0; i < size; i++) {

Res.val[i] = val[i] + num;

}

return Res;

};

TMatrix operator-(Type num) {

TMatrix Res(size);

for (int i = 0; i < size; i++) {

Res.val[i] = val[i] - num;

}

return Res;

};

TMatrix operator\*(Type num) {

TMatrix Res(size);

for (int i = 0; i < size; i++) {

Res.val[i] = val[i] \* num;

}

return Res;

};

TMatrix operator\*(const TMatrix& mat) {

TMatrix result(size);

if (size != mat.size) {

cout << "Несовместимые размеры матриц для умножения. Возвращена нулевая матрица." << endl;

}

else {

for (int i = 0; i < size; i++) {

for (int j = 0; j < size; j++) {

Type sum = 0;

for (int k = 0; k < size; k++) {

sum += val[i][k] \* mat.val[k][j];

}

result.val[i][j] = sum;

}

}

}

return result;

};

TVector<Type>& operator[](int index) {

while (index < 0 || index >= size) {

cout << "Ошибка: попытка обращения к недопустимому индексу матрицы.\n Введите корректное значение." << endl;

cin >> index;

}

return val[index];

};

void Input() {

for (int i = 0; i < size; i++) {

val[i].Input();

}

};

void Print() {

cout << endl;

for (int i = 0; i < size; i++) {

val[i].Print();

}

};

};