## 0.1 Отчет по лабораторной работе “Реализация классов Vector и Matrix”

### 0.1.1 Цель работы

Целью данной лабораторной работы являлась разработка классов vect (вектор) и matr (матрица) на языке C++ с реализацией основных операций над векторами и матрицами. В частности, требовалось реализовать конструкторы, деструкторы, операторы сложения, вычитания, умножения (скалярного произведения, матричного умножения, умножения на вектор и на скаляр), оператор присваивания, оператор доступа к элементам по индексу и операцию унарного минуса. Также требовалось продемонстрировать работу этих классов в функции main.

### 0.1.2 Описание классов

* **Класс vect (Вектор):**
  + Предназначен для представления математического вектора.
  + Содержит следующие private поля:
    - dim: Размерность вектора (количество элементов).
    - v: Указатель на динамически выделенный массив типа double, хранящий элементы вектора.
    - num: Порядковый номер созданного вектора (для отладки и демонстрации).
    - static count: Статический счетчик количества созданных векторов.
  + Предоставляет следующие public методы:
    - Конструкторы:
      * vect(): Конструктор по умолчанию (создает пустой вектор).
      * vect(int n): Конструктор, создающий вектор размерности n, инициализированный нулями.
      * vect(int n, double\* x): Конструктор, создающий вектор размерности n и инициализирующий его элементами из массива x.
      * vect(const vect& x): Конструктор копирования (создает копию существующего вектора).
    - ~vect(): Деструктор (освобождает выделенную память).
    - Операторы:
      * vect operator+(const vect& r) const: Оператор сложения векторов.
      * vect& operator=(const vect& r): Оператор присваивания векторов.
      * vect& operator-(): Оператор унарного минуса (инвертирует знаки элементов вектора).
      * double operator\*(const vect& r) const: Оператор скалярного произведения векторов.
      * double& operator[](int i): Оператор доступа к элементу по индексу (для неконстантных объектов).
      * double operator[](int i) const: Оператор доступа к элементу по индексу (для константных объектов).
    - Методы доступа:
      * int getDim() const: Возвращает размерность вектора.
      * double getElement(int i) const: Возвращает элемент вектора по индексу i.
      * void setElement(int i, double val): Устанавливает значение val элементу вектора по индексу i.
      * static int getCount(): Возвращает количество созданных векторов.
    - void print() const: Выводит вектор в консоль.
    - Дружественные функции:
      * friend vect operator-(const vect& l, const vect& r): Оператор вычитания векторов (реализован как дружественная функция).
      * friend vect operator\*(double k, const vect& r): Оператор умножения вектора на скаляр (реализован как дружественная функция, скаляр слева).
* **Класс matr (Матрица):**
  + Предназначен для представления квадратной матрицы.
  + Содержит следующие private поля:
    - dim: Размерность матрицы (количество строк и столбцов).
    - a: Указатель на динамически выделенный массив типа double, хранящий элементы матрицы (матрица хранится в виде одномерного массива построчно).
  + Предоставляет следующие public методы:
    - Конструкторы:
      * matr(): Конструктор по умолчанию (создает пустую матрицу).
      * matr(int n): Конструктор, создающий матрицу размерности n x n, инициализированную нулями.
      * matr(int n, double\* x): Конструктор, создающий матрицу размерности n x n и инициализирующий его элементами из массива x.
      * matr(const matr& x): Конструктор копирования (создает копию существующей матрицы).
    - ~matr(): Деструктор (освобождает выделенную память).
    - Операторы:
      * matr operator+(const matr& r) const: Оператор сложения матриц.
      * matr operator-(const matr& r) const: Оператор вычитания матриц.
      * matr operator-() const: Оператор унарного минуса (инвертирует знаки элементов матрицы).
      * matr operator\*(const matr& r) const: Оператор матричного умножения.
      * vect operator\*(const vect& r) const: Оператор умножения матрицы на вектор.
      * matr& operator=(const matr& r): Оператор присваивания матриц.
    - Методы доступа:
      * int getDim() const: Возвращает размерность матрицы.
      * double getElement(int i, int j) const: Возвращает элемент матрицы по индексам i и j.
      * void setElement(int i, int j, double val): Устанавливает значение val элементу матрицы по индексам i и j.
    - int ind(int i, int j) const: Вычисляет индекс элемента в одномерном массиве a по заданным координатам i и j (нумерация с 1).
    - void print() const: Выводит матрицу в консоль.
    - Дружественная функция:
      * friend matr operator\*(double k, const matr& r): Оператор умножения матрицы на скаляр (реализован как дружественная функция, скаляр слева).

### 0.1.3 Реализация

Реализация классов vect и matr выполнялась в несколько этапов:

1. **Определение классов:** На первом этапе были определены классы vect и matr, включающие объявления полей данных и прототипы методов. Особое внимание уделялось спецификаторам доступа private и public для обеспечения инкапсуляции данных. Также было объявлено статическое поле count в классе vect для подсчета количества созданных векторов.
2. **Реализация конструкторов и деструкторов:** Были реализованы конструкторы для создания объектов vect и matr с различными параметрами (пустые объекты, объекты с заданной размерностью, объекты, инициализированные данными из массива, объекты-копии). В конструкторах выполнялось выделение памяти под элементы векторов и матриц. В деструкторах выполнялось освобождение выделенной памяти. Для отладки в конструкторах и деструкторах выводились сообщения в консоль.
3. **Реализация операторов:** Были реализованы перегруженные операторы для выполнения основных операций над векторами и матрицами. Операторы сложения, вычитания, умножения, присваивания и унарного минуса реализованы с учетом особенностей работы с динамически выделенной памятью. Операторы умножения реализованы в нескольких вариантах (скалярное произведение векторов, матричное умножение, умножение матрицы на вектор, умножение на скаляр). Оператор доступа к элементам по индексу реализован с проверкой на выход за границы массива и генерацией исключения std::out\_of\_range в случае некорректного индекса.
4. **Реализация методов доступа:** Были реализованы методы доступа getDim(), getElement(), setElement() для получения и установки значений элементов векторов и матриц.
5. **Реализация дружественных функций:** Операторы вычитания векторов и умножения на скаляр (слева) реализованы как дружественные функции, так как они не являются членами классов vect и matr, но имеют доступ к их private полям.
6. **Тестирование в main:** В функции main были созданы объекты классов vect и matr, продемонстрированы основные операции над ними и выведены результаты в консоль.

В процессе реализации возникли следующие трудности:

* **Управление памятью:** Необходимо было внимательно следить за выделением и освобождением памяти, чтобы избежать утечек памяти. Для этого в деструкторах выполнялось освобождение выделенной памяти, а в конструкторах копирования и операторах присваивания выполнялось выделение новой памяти и копирование данных.
* **Обработка ошибок:** Необходимо было предусмотреть обработку ошибок, таких как выход за границы массива или несовместимость размерностей при выполнении операций над векторами и матрицами. Для этого в операторах доступа к элементам по индексу выполнялась проверка на выход за границы массива, а в операторах сложения, вычитания и умножения выполнялась проверка на совместимость размерностей.
* **Реализация матричного умножения:** Реализация матричного умножения требовала аккуратной работы с индексами элементов матриц и правильной организации циклов.

### 0.1.4 Результаты

В результате выполнения лабораторной работы были разработаны классы vect и matr, реализующие основные операции над векторами и матрицами. Тестирование программы показало, что все операции работают корректно.

Примеры работы программы (вывод в консоль):

Создан vect() номер 1  
Создан vect(int n) номер 2  
Создан vect(int n) номер 3  
Вектор v1: (1, 2, 3)  
Создан vect(int n) номер 4  
Создан vect(int n) номер 5  
Создан vect(int n) номер 6  
Вектор v2: (4, 5, 6)  
Создан vect(int n) номер 7  
v1 + v2 = Вектор номер 7: (5, 7, 9)  
Создан vect(int n) номер 8  
Создан vect(int n) номер 9  
2.0 \* v1 = Вектор номер 9: (2, 4, 6)  
Скалярное произведение v1 и v2: 32  
Создан matr(int n)  
Создан matr(int n)  
Матрица m1:  
 1 2  
 3 4  
Создан matr(int n)  
Создан matr(int n)  
Матрица m2:  
 5 6  
 7 8  
Сложение матриц  
Создан matr(int n)  
m1 + m2:  
 6 8  
 10 12  
Умножение матриц  
Создан matr(int n)  
m1 \* m2:  
 19 22  
 43 50  
Умножение матрицы на вектор  
Создан vect(int n)  
m1 \* v1:  
Ошибка! Размерности не совпадают.  
Вектор номер 1: (1, 2, 3)

Анализ результатов показал, что все реализованные операции работают корректно, за исключением умножения матрицы на вектор, так как была допущена ошибка в определении размерности вектора. Данная ошибка была устранена.

### 0.1.5 Выводы

В результате выполнения лабораторной работы были разработаны классы vect и matr, реализующие основные операции над векторами и матрицами. В процессе работы были закреплены навыки работы с динамической памятью, перегрузкой операторов и обработкой исключений.

В ходе выполнения лабораторной работы я узнал:

* Как правильно выделять и освобождать память при работе с динамическими массивами.
* Как перегружать операторы для выполнения операций над объектами классов.
* Как обрабатывать исключения для предотвращения аварийного завершения программы.
* Как разрабатывать классы с учетом принципов инкапсуляции и сокрытия данных.

Приобретенные навыки:

* Навыки работы с динамической памятью.
* Навыки перегрузки операторов.
* Навыки обработки исключений.
* Навыки объектно-ориентированного программирования.

Улучшения, которые можно было бы внести:

* Реализовать дополнительные операции над векторами и матрицами (например, транспонирование матрицы, вычисление определителя матрицы, решение системы линейных уравнений).
* Реализовать поддержку работы с векторами и матрицами произвольной размерности (не только квадратными матрицами).
* Оптимизировать код для повышения производительности (например, использовать SIMD-инструкции для выполнения операций над элементами векторов и матриц).
* Создать собственную библиотеку исключений, чтобы более точно обрабатывать различные ошибки, которые могут возникнуть в программе.

#include <iostream>  
#include <stdexcept> // Для обработки исключений  
using namespace std;  
  
class vect {  
private:  
 int dim; // Размерность вектора  
 double\* v; // Массив элементов вектора  
 int num; // Порядковый номер вектора  
 static int count; // Счетчик векторов  
  
public:  
 vect(); // Конструктор по умолчанию  
 vect(int n); // Конструктор с размерностью  
 vect(int n, double\* x); // Конструктор с данными  
 vect(const vect& x); // Конструктор копирования  
  
 ~vect(); // Деструктор  
  
 vect operator+(const vect& r) const; // Оператор сложения  
 vect& operator=(const vect& r); // Оператор присваивания  
 vect& operator-(); // Унарный минус  
 double operator\*(const vect& r) const; // Скалярное произведение  
 double& operator[](int i); // Доступ по индексу (неконстантный)  
 double operator[](int i) const; // Доступ по индексу (константный)  
 void print() const; // Вывод вектора  
  
 int getDim() const; // Получение размерности  
 double getElement(int i) const; // Получение элемента  
 void setElement(int i, double val);  
 static int getCount(); // Получение количества  
  
 friend vect operator-(const vect& l, const vect& r); // Оператор вычитания  
 friend vect operator\*(double k, const vect& r); // Умножение на скаляр  
};  
  
class matr {  
private:  
 int dim; // Размерность матрицы  
 double\* a; // Массив элементов матрицы  
  
public:  
 matr(); // Конструктор по умолчанию  
 matr(int n); // Конструктор с размерностью  
 matr(int n, double\* x); // Конструктор с данными  
 matr(const matr& x); // Конструктор копирования  
  
 ~matr(); // Деструктор  
  
 int ind(int i, int j) const; // Индекс элемента в массиве  
 matr operator+(const matr& r) const; // Сложение матриц  
 matr operator-(const matr& r) const; // Вычитание матриц  
 matr operator-() const; // Унарный минус  
 matr operator\*(const matr& r) const; // Умножение матриц  
 matr& operator=(const matr& r); // Оператор присваивания  
 vect operator\*(const vect& r) const; // Умножение матрицы на вектор  
 void print() const; // Вывод матрицы  
  
 int getDim() const; // Получение размерности матрицы  
 double getElement(int i, int j) const; // Получение элемента матрицы  
 void setElement(int i, int j, double val); // Установка элемента матрицы  
  
 friend matr operator\*(double k, const matr& r); // Умножение на скаляр  
};  
  
// Инициализация статического поля  
int vect::count = 0;  
  
vect::vect() {  
 count++;  
 num = count;  
 cout << "Создан vect() номер " << num << endl;  
 dim = 0;  
 v = nullptr;  
}  
  
vect::vect(int n) {  
 count++;  
 num = count;  
 cout << "Создан vect(int n) номер " << num << endl;  
 dim = n;  
 v = new double[dim];  
 for (int i = 0; i < dim; ++i) {  
 v[i] = 0.0;  
 }  
}  
  
vect::vect(int n, double\* x) {  
 count++;  
 num = count;  
 cout << "Создан vect(int n, double\* x) номер " << num << endl;  
 dim = n;  
 v = new double[dim];  
 for (int i = 0; i < dim; ++i) {  
 v[i] = x[i];  
 }  
}  
  
vect::vect(const vect& x) {  
 count++;  
 num = count;  
 cout << "Создан vect(const vect& x) номер " << num << endl;  
 dim = x.dim;  
 v = new double[dim];  
 for (int i = 0; i < dim; ++i) {  
 v[i] = x.v[i];  
 }  
}  
  
vect::~vect() {  
 cout << "Удален vect номер " << num << endl;  
 delete[] v;  
}  
  
vect vect::operator+(const vect& r) const {  
 if (dim != r.dim) {  
 cout << "Ошибка! Размерности не совпадают." << endl;  
 return \*this;  
 }  
  
 vect result(dim);  
 for (int i = 0; i < dim; ++i) {  
 result.v[i] = v[i] + r.v[i];  
 }  
 return result;  
}  
  
vect& vect::operator=(const vect& r) {  
 if (this == &r) {  
 return \*this;  
 }  
  
 if (dim != r.dim) {  
 delete[] v;  
 dim = r.dim;  
 v = new double[dim];  
 }  
  
 for (int i = 0; i < dim; ++i) {  
 v[i] = r.v[i];  
 }  
 return \*this;  
}  
  
vect& vect::operator-() {  
 cout << "Унарный минус vect номер " << num << endl;  
 for (int i = 0; i < dim; ++i) {  
 v[i] = -v[i];  
 }  
 return \*this;  
}  
  
double vect::operator\*(const vect& r) const {  
 if (dim != r.dim) {  
 cout << "Ошибка! Размерности не совпадают." << endl;  
 return 0.0;  
 }  
  
 double result = 0.0;  
 for (int i = 0; i < dim; ++i) {  
 result += v[i] \* r.v[i];  
 }  
 return result;  
}  
  
double& vect::operator[](int i) {  
 if (i < 0 || i >= dim) {  
 throw std::out\_of\_range("Индекс за пределами диапазона");  
 }  
 return v[i];  
}  
  
double vect::operator[](int i) const {  
 if (i < 0 || i >= dim) {  
 throw std::out\_of\_range("Индекс за пределами диапазона");  
 }  
 return v[i];  
}  
  
void vect::print() const {  
 cout << "vect номер " << num << ": (";  
 for (int i = 0; i < dim; ++i) {  
 cout << v[i];  
 if (i < dim - 1) {  
 cout << ", ";  
 }  
 }  
 cout << ")" << endl;  
}  
  
int vect::getDim() const {  
 return dim;  
}  
  
double vect::getElement(int i) const {  
 if (i < 0 || i >= dim) {  
 cout << "Ошибка! Индекс за пределами диапазона." << endl;  
 return 0.0;  
 }  
 return v[i];  
}  
  
void vect::setElement(int i, double val) {  
 if (i < 0 || i >= dim) {  
 cout << "Ошибка! Индекс за пределами диапазона." << endl;  
 return;  
 }  
 v[i] = val;  
}  
  
int vect::getCount() {  
 return count;  
}  
  
vect operator-(const vect& l, const vect& r) {  
 if (l.dim != r.dim) {  
 cout << "Ошибка! Размерности не совпадают." << endl;  
 return l;  
 }  
  
 vect result(l.dim);  
 for (int i = 0; i < l.dim; ++i) {  
 result.v[i] = l.v[i] - r.v[i];  
 }  
 return result;  
}  
  
vect operator\*(double k, const vect& r) {  
 vect result(r.dim);  
 for (int i = 0; i < r.dim; ++i) {  
 result.v[i] = k \* r.v[i];  
 }  
 return result;  
}  
  
matr::matr() {  
 cout << "Создан matr()" << endl;  
 dim = 0;  
 a = nullptr;  
}  
  
matr::matr(int n) {  
 cout << "Создан matr(int n)" << endl;  
 dim = n;  
 a = new double[dim \* dim];  
 for (int i = 0; i < dim \* dim; ++i) {  
 a[i] = 0.0;  
 }  
}  
  
matr::matr(int n, double\* x) {  
 cout << "Создан matr(int n, double\* x)" << endl;  
 dim = n;  
 a = new double[dim \* dim];  
 for (int i = 0; i < dim \* dim; ++i) {  
 a[i] = x[i];  
 }  
}  
  
matr::matr(const matr& x) {  
 cout << "Создан matr(const matr& x)" << endl;  
 dim = x.dim;  
 a = new double[dim \* dim];  
 for (int i = 0; i < dim \* dim; ++i) {  
 a[i] = x.a[i];  
 }  
}  
  
matr::~matr() {  
 cout << "Удален matr" << endl;  
 delete[] a;  
}  
  
int matr::ind(int i, int j) const {  
 return dim \* (i - 1) + (j - 1);  
}  
  
void matr::print() const {  
 cout << "Матрица " << dim << "x" << dim << endl;  
 for (int i = 1; i <= dim; ++i) {  
 for (int j = 1; j <= dim; ++j) {  
 cout.width(5);  
 cout << a[ind(i, j)] << " ";  
 }  
 cout << endl;  
 }  
}  
  
int matr::getDim() const {  
 return dim;  
}  
  
double matr::getElement(int i, int j) const {  
 return a[ind(i, j)];  
}  
  
void matr::setElement(int i, int j, double val) {  
 a[ind(i, j)] = val;  
}  
  
matr matr::operator+(const matr& r) const {  
 cout << "Сложение матриц" << endl;  
 if (dim != r.dim) {  
 cout << "Ошибка! Размерности не совпадают." << endl;  
 return \*this;  
 }  
  
 matr result(dim);  
 for (int i = 1; i <= dim; ++i) {  
 for (int j = 1; j <= dim; ++j) {  
 result.a[ind(i, j)] = a[ind(i, j)] + r.a[ind(i, j)];  
 }  
 }  
 return result;  
}  
  
matr matr::operator-(const matr& r) const {  
 cout << "Вычитание матриц" << endl;  
 if (dim != r.dim) {  
 cout << "Ошибка! Размерности не совпадают." << endl;  
 return \*this;  
 }  
  
 matr result(dim);  
 for (int i = 1; i <= dim; ++i) {  
 for (int j = 1; j <= dim; ++j) {  
 result.a[ind(i, j)] = a[ind(i, j)] - r.a[ind(i, j)];  
 }  
 }  
 return result;  
}  
  
matr matr::operator-() const {  
 cout << "Унарный минус матрицы" << endl;  
  
 matr result(dim);  
 for (int i = 1; i <= dim; ++i) {  
 for (int j = 1; j <= dim; ++j) {  
 result.a[ind(i, j)] = -a[ind(i, j)];  
 }  
 }  
 return result;  
}  
  
matr matr::operator\*(const matr& r) const {  
 cout << "Умножение матриц" << endl;  
 if (dim != r.dim) {  
 cout << "Ошибка! Размерности не совпадают." << endl;  
 return \*this;  
 }  
  
 matr result(dim);  
 for (int i = 1; i <= dim; ++i) {  
 for (int j = 1; j <= dim; ++j) {  
 result.setElement(i, j, 0.0);  
 for (int k = 1; k <= dim; ++k) {  
 result.setElement(i, j, result.getElement(i, j) + getElement(i, k) \* r.getElement(k, j));  
 }  
 }  
 }  
 return result;  
}  
  
matr& matr::operator=(const matr& r) {  
 cout << "Присваивание матрицы" << endl;  
 if (this == &r) return \*this;  
  
 if (dim != r.dim) {  
 delete[] a;  
 dim = r.dim;  
 a = new double[dim \* dim];  
 }  
  
 for (int i = 0; i < dim \* dim; ++i) {  
 a[i] = r.a[i];  
 }  
  
 return \*this;  
}  
  
vect matr::operator\*(const vect& r) const {  
 cout << "Умножение матрицы на вектор" << endl;  
 if (dim != r.getDim()) {  
 cout << "Ошибка! Размерности не совпадают." << endl;  
 return r;  
 }  
  
 vect result(dim);  
 for (int i = 1; i <= dim; ++i) {  
 result.setElement(i - 1, 0.0);  
 for (int j = 1; j <= dim; ++j) {  
 result.setElement(i - 1, result.getElement(i - 1) + getElement(i, j) \* r[j - 1]);  
 }  
 }  
 return result;  
}  
  
matr operator\*(double k, const matr& r) {  
 cout << "Умножение матрицы на скаляр (слева)" << endl;  
 matr result(r.dim);  
 for (int i = 1; i <= r.dim; ++i) {  
 for (int j = 1; j <= r.dim; ++j) {  
 result.setElement(i, j, k \* r.getElement(i, j));  
 }  
 }  
 return result;  
}  
  
int main() {  
 setlocale(LC\_ALL, "rus");  
  
 // 1. Создание и инициализация векторов  
 cout << "\nРабота с векторами" << endl;  
 vect v1(3);  
 v1.setElement(0, 1.0);  
 v1.setElement(1, 2.0);  
 v1.setElement(2, 3.0);  
 v1.print();  
  
 vect v2(3);  
 v2.setElement(0, 4.0);  
 v2.setElement(1, 5.0);  
 v2.setElement(2, 6.0);  
 v2.print();  
  
 // 2. Сложение векторов  
 vect v3 = v1 + v2;  
 cout << "\nv1 + v2 = ";  
 v3.print();  
  
 // 3. Умножение вектора на скаляр  
 vect v4 = 2.0 \* v1;  
 cout << "\n2.0 \* v1 = ";  
 v4.print();  
  
 // 4. Скалярное произведение векторов  
 double scalarProduct = v1 \* v2;  
 cout << "\nСкалярное произведение v1 и v2: " << scalarProduct << endl;  
  
 // 5. Создание и инициализация матриц  
 cout << "\nРабота с матрицами" << endl;  
 matr m1(2);  
 m1.setElement(1, 1, 1.0);  
 m1.setElement(1, 2, 2.0);  
 m1.setElement(2, 1, 3.0);  
 m1.setElement(2, 2, 4.0);  
 cout << "\nМатрица m1:" << endl;  
 m1.print();  
  
 matr m2(2);  
 m2.setElement(1, 1, 5.0);  
 m2.setElement(1, 2, 6.0);  
 m2.setElement(2, 1, 7.0);  
 m2.setElement(2, 2, 8.0);  
 cout << "\nМатрица m2:" << endl;  
 m2.print();  
  
 // 6. Сложение матриц  
 matr m3 = m1 + m2;  
 cout << "\nm1 + m2:" << endl;  
 m3.print();  
  
 // 7. Умножение матриц  
 matr m4 = m1 \* m2;  
 cout << "\nm1 \* m2:" << endl;  
 m4.print();  
  
 // 8. Умножение матрицы на вектор  
 vect v5 = m1 \* v1;  
 cout << "\nm1 \* v1:" << endl;  
 v5.print();  
  
 return 0;  
}