**МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ**

**НОВОСИБИРСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСТИЕТ**

Лабораторная работа №1 по дисциплине «Программирование»

**Разработка классов, создание конструкторов и деструкторов. Использование статических членов класса**

Группа: **АВТ-342**

Студенты: **Бондаренко А.В.,Фадеев В.А.**

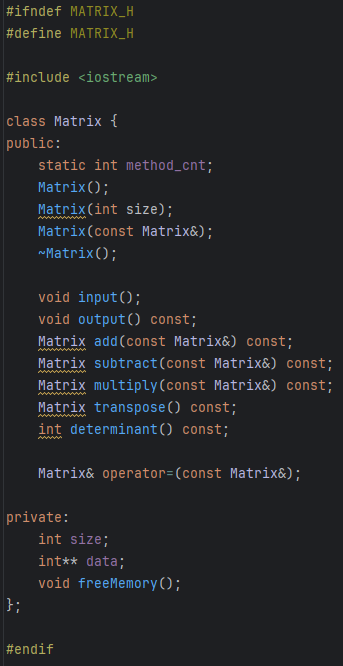
НОВОСИБИРСК 2024

**1.Постановка задачи**

**Вариант 4.**

Постpоить класс для pаботы с квадратными матрицами. Класс должен включать соответствующие поля: порядок матрицы, набор элементов. Класс должен обеспечивать пpостейшие функции для pаботы с данными класса: транспонирование матрицы, сложение матриц, вывод матрицы в удобной форме на экран и т.д.

**2. Определение пользовательского класса**

Создадим файл matrix.h, где у нас будет находиться пользовательский класс с методами в нём.

method\_cnt – статический член, который будет подсчитывать количество задействованных меетодов при выполнении программы.  
Matrix() – конструктор по умолчанию.

Matrix(int size) – конструктор с параметром.

Matrix(const Matrix&) – конструктор копирования.  
~Matrix() – деструктор.

Input – метод для ввода матрицы. Так же в нём проверяется, что вводится именно число.

Output – метод для вывода матрицы.

Add – метод для сложения матриц.

Subtract – метод для вычитания матриц.

Multiply – метод для умножения матриц.

Transpose – метод для транспонирования матриц.

Determinant – метод для вычисления определителя матрицы.

Matrix& operator=(const Matrix&) – оператор присваивания.

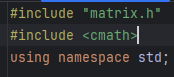
Переменная size – размер матрицы.

Int\*\* data – матрица.

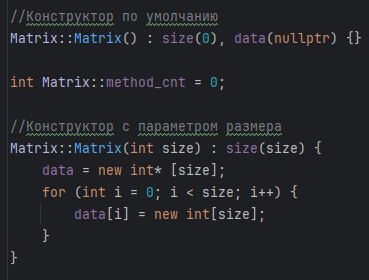
Freememory – метод для очистки памяти.

**3. Листинг программы**

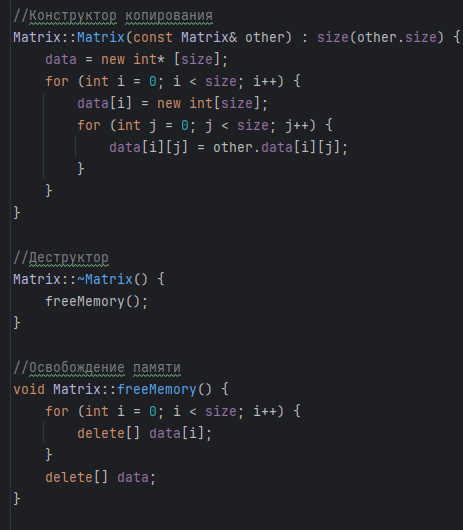
Далее создадим файл matrix.cpp, где опишем методы.

Начнём с библиотек:  


Matrix.h нужна для того, чтобы можно было из matrix.cpp использовать методы. Cmath пригодится для определителя.



Создаём конструктор по умолчанию, задаём начальное значение для счётчика модулей, задаём конструктор с параметром размера.



Пишем конструктор копирования (он нам в дальнейшем не пригодится, но он является обязательным для данной лабараторной работы).

Деструктор – для очистки памяти.

Freememory – сам алгоритм очистки, в данном случае динамической.



При вызове метода input выводится сообщение с просьбой ввести элементы матрицы размера size x size. Это информирует пользователя о том, сколько элементов нужно ввести.

Двумерный цикл для ввода данных: Используются два вложенных цикла for, которые проходят по каждой строке и каждому столбцу матрицы. Внешний цикл проходит по строкам матрицы (индекс i), а внутренний — по столбцам (индекс j).

Ввод значений: Для каждой ячейки матрицы ожидается ввод целочисленного значения. Внутри цикла выполняется бесконечный цикл while (true), который контролирует ввод данных.

Проверка ввода: Ввод данных проверяется с помощью механизма обнаружения ошибок в потоке ввода:

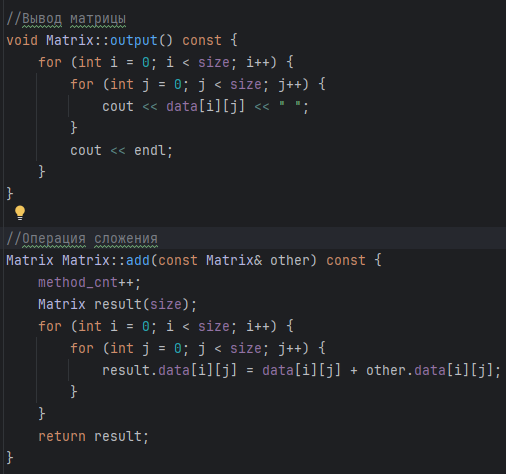
◦ Если ввод некорректен (например, пользователь ввёл не число), то cin.fail() вернёт true. Это указывает на ошибку ввода.

◦ В таком случае поток ввода очищается с помощью cin.clear(), чтобы сбросить флаг ошибки.

◦ Затем с помощью cin.ignore() игнорируется оставшаяся часть строки ввода, и программа выводит сообщение "n/a", после чего завершается с ошибкой с помощью exit(1).

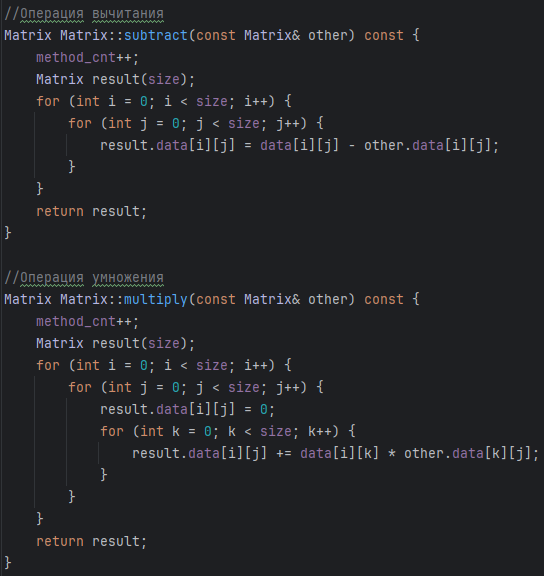
5 Успешный ввод: Если ввод корректен, введённое значение сохраняется в ячейку матрицы data[i][j], после чего выполнение продолжается для следующего элемента.

Таким образом, программа защищена от некорректного ввода и завершает работу с ошибкой, если пользователь ввёл что-то, кроме целого числа.



Output идёт по i и j координатам матрицы и выводит каждый элемент матрицы с пробелами между ними. При окончании строки, ввод переносится на следующую строку.

Сложение так же работает через двумерные массивы, где каждый [i][j] элемент складывается с [i][j] элементом второй матрицы.



Вычитание работает аналогично сложению, но вместо “+” здесь “-”.

Тройной цикл для умножения матриц:

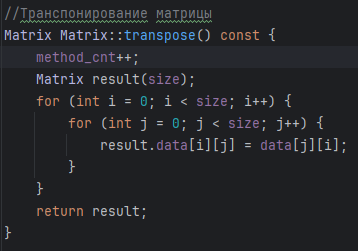
◦ Первый цикл (for (int i = 0; i < size; i++)) отвечает за строки результирующей матрицы.

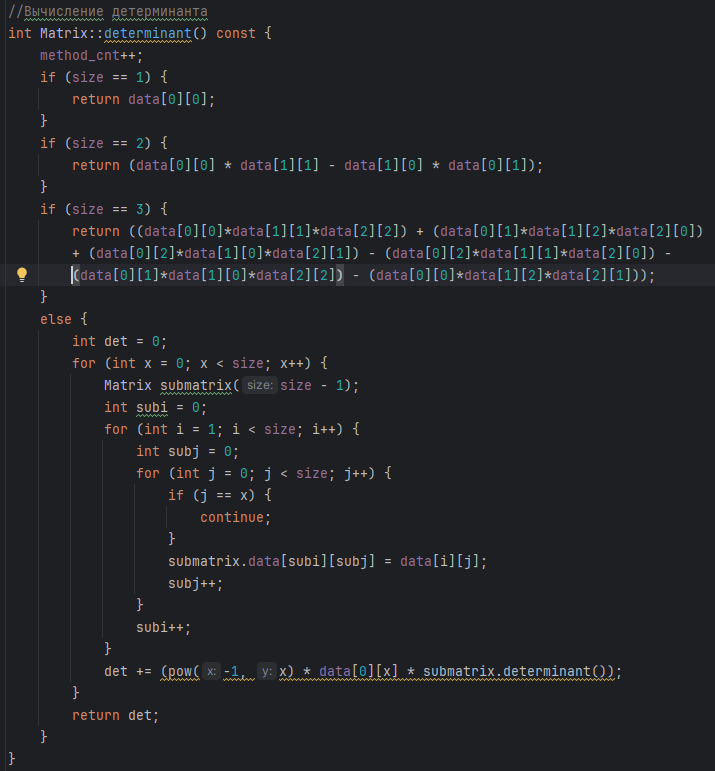
◦ Второй цикл (for (int j = 0; j < size; j++)) отвечает за столбцы результирующей матрицы.

◦ Инициализация: Каждая ячейка результирующей матрицы result.data[i][j] инициализируется нулём перед тем, как в неё будет записываться результат умножения.

◦ Третий цикл (for (int k = 0; k < size; k++)) отвечает за вычисление суммы произведений соответствующих элементов строк первой матрицы и столбцов второй матрицы. Это реализует стандартное правило умножения матриц, при котором элемент результирующей матрицы на позиции [i][j] равен сумме произведений элементов i-й строки первой матрицы и j-го столбца второй матрицы.

2 Возврат результата: После завершения вычислений метод возвращает объект result, который представляет собой результат умножения двух матриц.



Транспонирование – к каждому элементу первой матриц приравнивается противоположный по координатам элемент второй матрицы.

1 Обработка матриц маленького размера:

◦ Если размер матрицы равен 1 (size == 1), метод возвращает единственный элемент матрицы, так как определитель 1x1 матрицы — это её единственное значение.

◦ Если размер матрицы равен 2 (size == 2), используется стандартная формула для вычисления определителя 2x2 матрицы.

◦ Если размер матрицы равен 3 (size == 3), применяется известная формула для определителя 3x3 матрицы, которая основана на комбинировании произведений диагональных элементов.

Рекурсивное вычисление для больших матриц: Если размер матрицы больше 3, используется разложение по строке:

◦ Инициализируется переменная det, которая будет хранить сумму для определителя.

◦ В цикле for (int x = 0; x < size; x++) происходит итерация по элементам первой строки матрицы (data[0][x]), для каждого элемента:

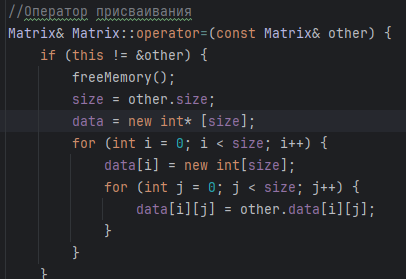
▪ Создаётся подматрица (минор) размера (size-1) x (size-1) для вычисления миноров, исключая строку 0 и столбец x.

▪ Сначала инициализируется индекс строки подматрицы subi, затем происходит заполнение подматрицы с исключением столбца x (используются индексы subi и subj для обхода исходной матрицы).

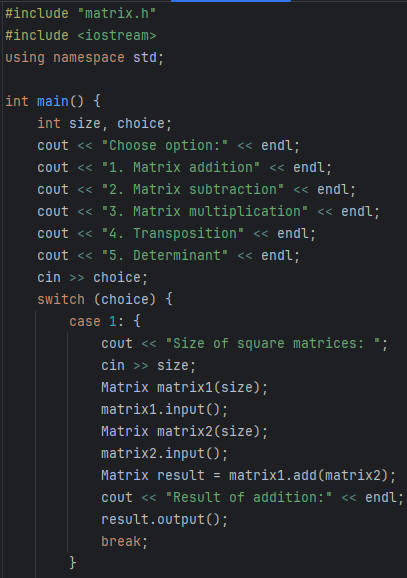
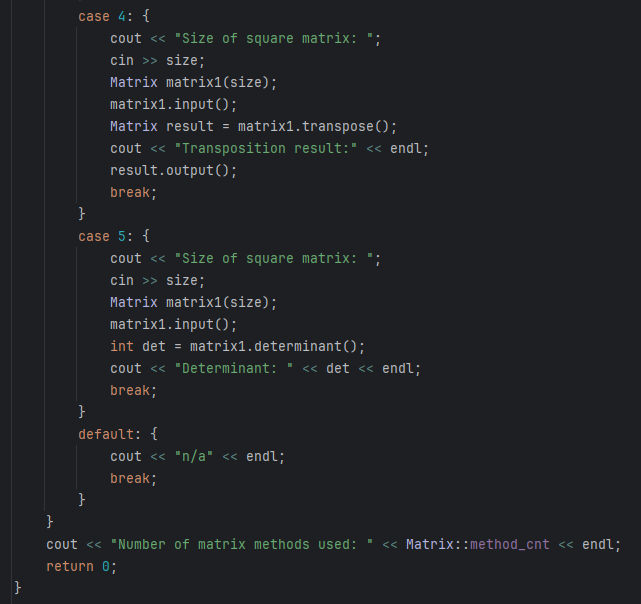
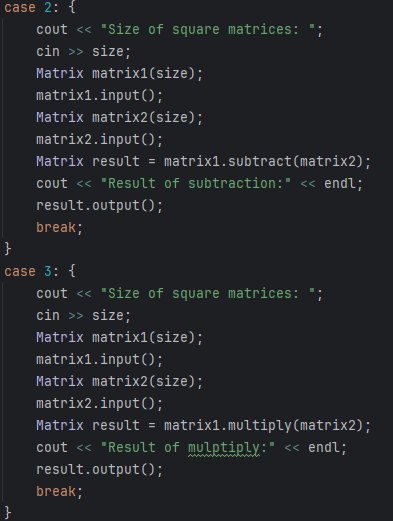
◦ Вызывается рекурсивный метод submatrix.determinant() для вычисления определителя подматрицы.

◦ Суммирование происходит с учётом знака: используется выражение pow(-1, x), которое чередует знаки для каждого элемента строки, согласно правилу разложения по строке:d

Возврат результата: Когда вся сумма рассчитана, метод возвращает значение определителя для исходной матрицы.



Оператор присваивания.

Далее создаём main.cpp, где перечислим доступные способы обработки матриц и создадим весь интерфейс программы.

Пользователь вводит значение переменной choice, в зависимости от которой выбирается тот метод, с которым будет работать пользователь. В самих case уже делается ввод матриц, обращение к методам и вывод.

После вывода результата работы с матрицами, выводится количество задействованных методов, он же – статический член.