МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКОЙ ФЕДЕРАЦИИ
федеральное государственное автономное образовательное учреждение высшего образования
"Омский государственный технический университет"

Кафедра «Автоматизированные системы обработки информации и управления»

ПРАКТИЧЕСКАЯ РАБОТА №2

по дисциплине

«Измерительные средства аналитики программных систем и технологий»

по теме

«Метрики Мак-Кейба.»

Выполнил: Шмидт А.В.
гр. ИВТ - 244
Проверил: Зубарев А.А.
к.т.н., доцент

Омск 2025

ЗАДАНИЕ

1. Выбрать программу, в которой ОБЯЗАТЕЛЬНО должны быть ВЕТВЛЕНИЯ, или ЦИКЛЫ (или то и другое). Представить графически схему алгоритма программы.
2. Представить поток управления в виде графа, сделав оценку свойств передачи управления в программе («дерево обязательного предшествования»).
3. Определить цикломатическую сложность графа потока управления.
4. Сделать выводы по работе.

ПРИМЕР ОПРЕДЕЛЕНИЯ МЕТРИК ПО МАК-КЕЙБУ

Программа на C++ генерирует две случайные целочисленные матрицы 7×7 (значения от −50 до 50). Для каждой матрицы выводит полностью матрицу, главную и побочную диагонали, а также их суммы. Дополнительно вычисляет и выводит разность: сумма главной диагонали первой матрицы минус сумма побочной диагонали второй матрицы. Ниже представлен листинг программы.

```
#include <iostream>
#include <iomanip>
#include <random>
#include <string>
using namespace std;

const int N = 7;

// Print main (true) or anti-diagonal (false)
void printDiagonal(const int matrix[N][N], bool isMain)
{
    for (int i = 0; i < N; ++i)
    {
        if (isMain)
            cout << setw(4) << matrix[i][i];
        else
            cout << setw(4) << matrix[i][N - 1 - i];
    }
    cout << "\n";
}

// Calculate sum of main (true) or anti-diagonal (false)
long long sumDiagonal(const int matrix[N][N], bool isMain)
{
    long long sum = 0;
    for (int i = 0; i < N; ++i)
    {
        sum += isMain ? matrix[i][i] : matrix[i][N - 1 - i];
    }
    return sum;
}

// Fill matrix with random integers in range [min, max]
void fillRandom(int matrix[N][N], int minVal = -50, int maxVal = 50)
{
    random_device rd;
    mt19937 gen(rd());
    uniform_int_distribution<> dis(minVal, maxVal);

    for (int i = 0; i < N; ++i)
        for (int j = 0; j < N; ++j)
            matrix[i][j] = dis(gen);
}
```

```
// Print full matrix
void printMatrix(const int matrix[N][N], const string& name)
{
    cout << "Matrix " << name << ":\n";
    for (int i = 0; i < N; ++i)
    {
        for (int j = 0; j < N; ++j)
            cout << setw(4) << matrix[i][j];
        cout << "\n";
    }
    cout << "\n";
}

// Process and display information for one matrix
void processMatrix(const int matrix[N][N], const string& name)
{
    cout << "Matrix " << name << ":\n";

    cout << "  Main diagonal:    ";
    printDiagonal(matrix, true);

    cout << "  Anti-diagonal:    ";
    printDiagonal(matrix, false);

    long long sumMain = sumDiagonal(matrix, true);
    long long sumAnti = sumDiagonal(matrix, false);

    cout << "  Sum of main diagonal:     " << sumMain << "\n";
    cout << "  Sum of anti-diagonal:     " << sumAnti << "\n\n";
}

int main()
{
    int A[N][N], B[N][N];

    // Seed is different each run thanks to random_device
    fillRandom(A);
    fillRandom(B);

    // Optional: print full matrices
    printMatrix(A, "A");
    printMatrix(B, "B");

    processMatrix(A, "A");
    processMatrix(B, "B");

    long long difference = sumDiagonal(A, true) - sumDiagonal(B, false);
    cout << "Difference (main diagonal of A - anti-diagonal of B) = " << difference << "\n";

    cin.get();
    return 0;
}
```

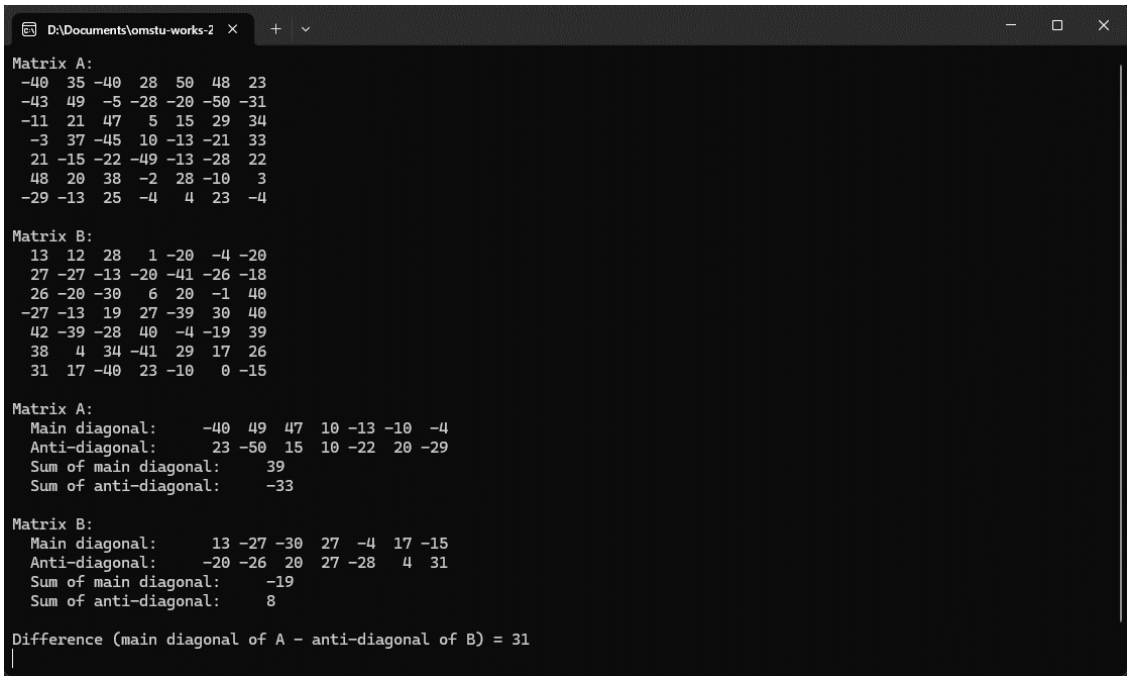

Рисунок 1 – Результат работы программы.

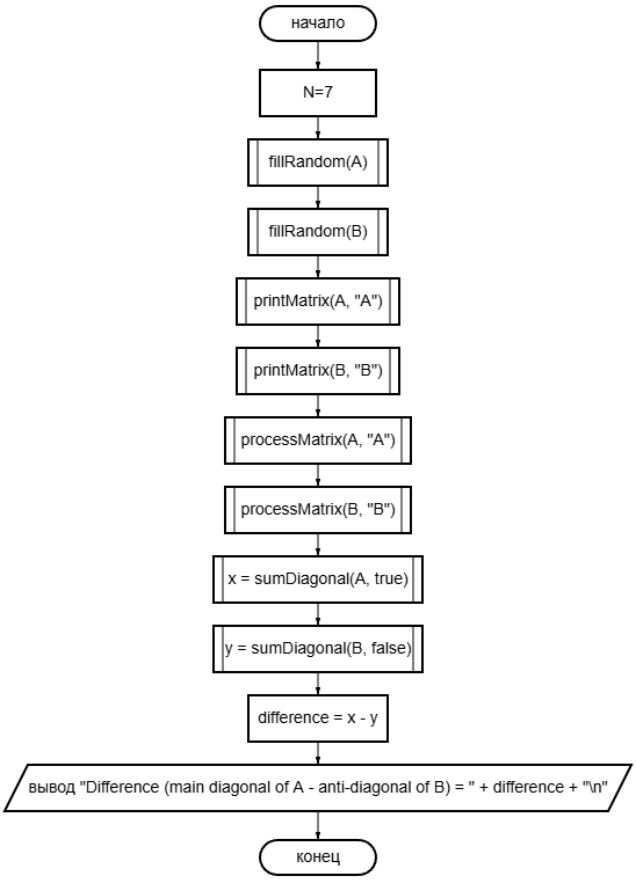

Рисунок 2 – Схема алгоритма главной функции.

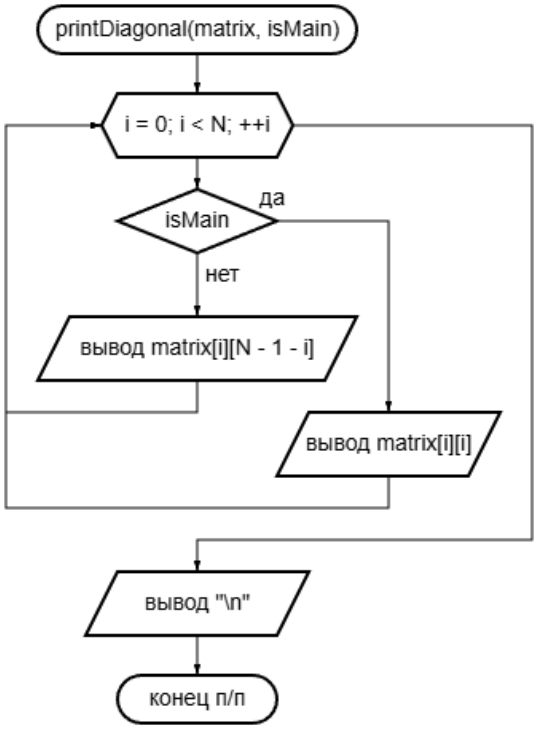

Рисунок 3 – Схема алгоритма подпрограммы для вывода элементов диагоналей матриц.

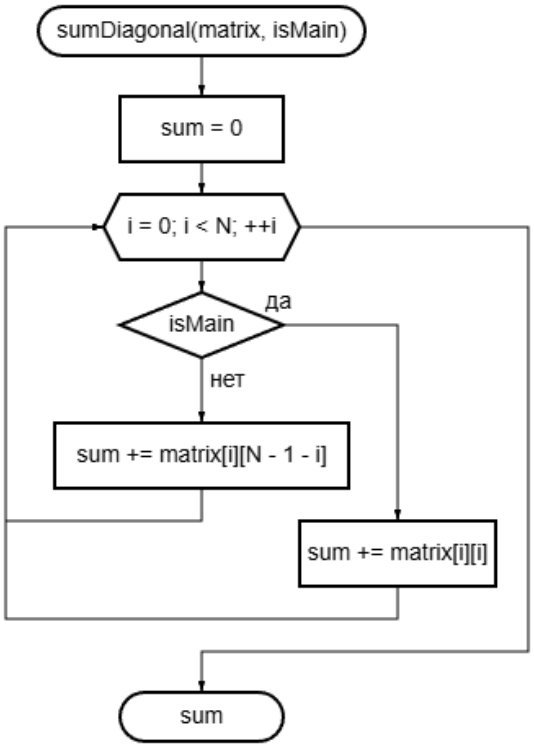

Рисунок 4 – Схема алгоритма подпрограммы, возвращающей сумму элементов диагонали матрицы.

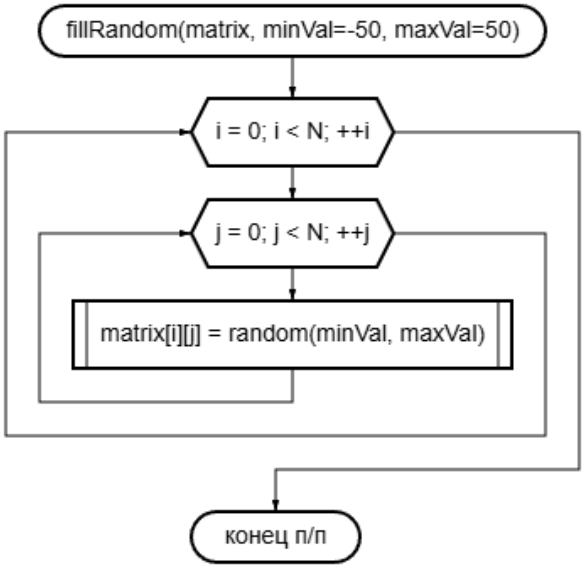

Рисунок 5 – Схема алгоритма подпрограммы для заполнения матрицы случайными числами.

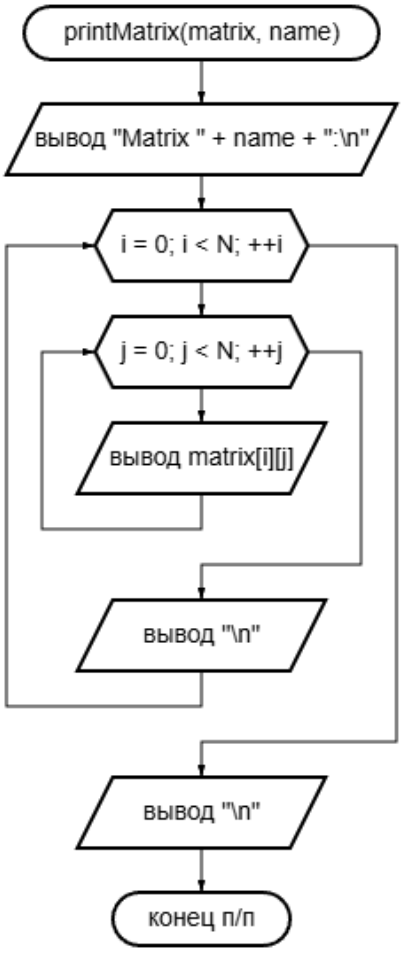

Рисунок 6 – Схема алгоритма подпрограммы для полного вывода матрицы.

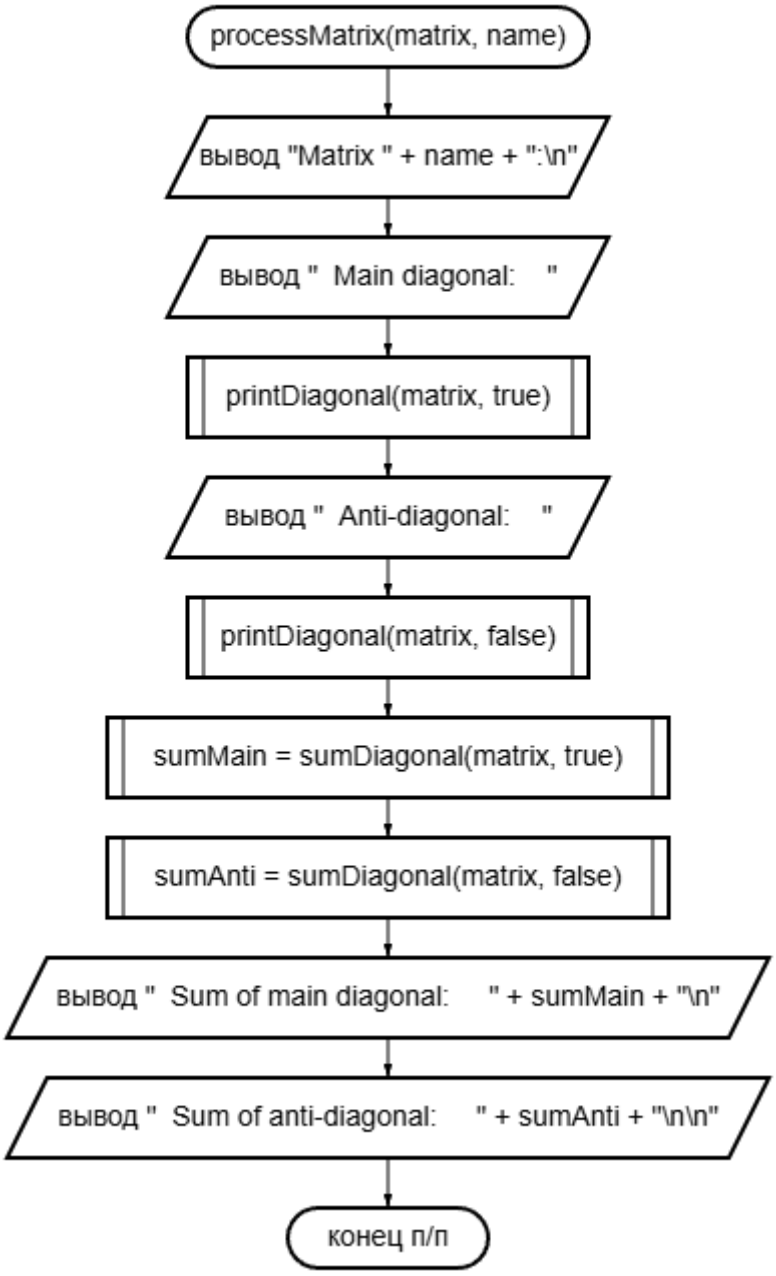

Рисунок 7 – Схема алгоритма подпрограммы для централизованной обработки и вывода матрицы.

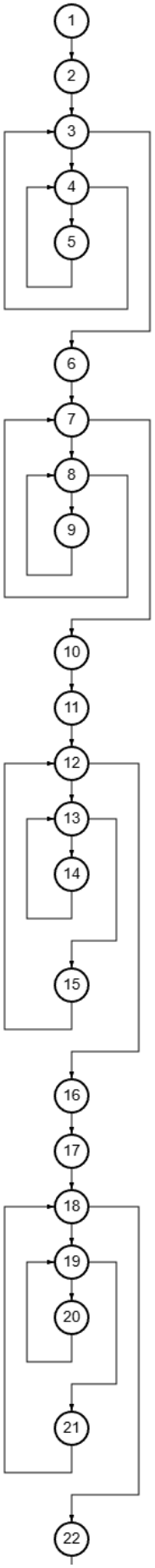

Рисунок 8 – Граф потока управления 1.

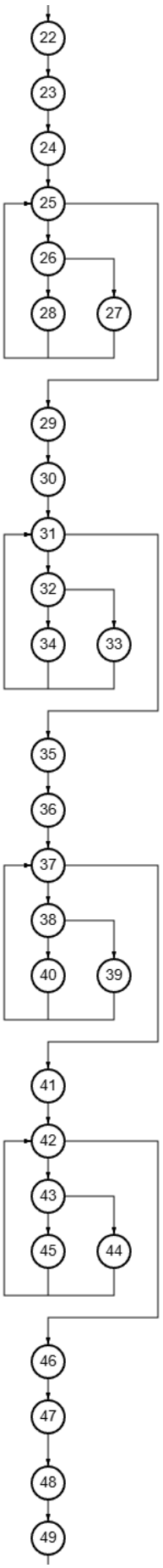

Рисунок 9 – Граф потока управления 2.

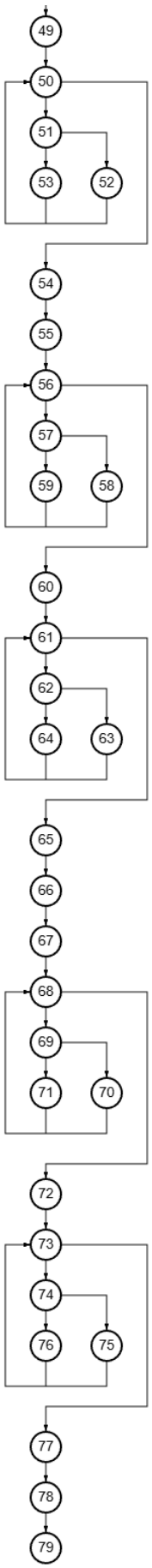

Рисунок 10 – Граф потока управления 3.

Таблица 1 – Таблица для подсчёта цикломатического числа.

Функция	Узлы (v)	Рёбра (e)	Примечания
fillRandom	7	10	2 вложенных for → 3 входа/выхода + 4 перехода внутри
printMatrix	7	10	аналогично fillRandom
printDiagonal	7	11	for + if → больше переходов
sumDiagonal	6	9	for + тернарный оператор
processMatrix	8	8	только последовательные вызовы + cout
main	7	6	только последовательные вызовы функций

Формула для вычисления цикломатического числа Мак-Кейба (цикломатической сложности) Z(G):

Z(G) = e - v + 2p, где:

e - количество рёбер в графе потока управления;

v – количество узлов (вершин);

p – количество связных компонент (здесь P = 1 — одна программа).

В нашем случае имеем:

Общее количество узлов v ≈ 7 + 7 + 7 + 6 + 8 + 7 = 42.

Общее количество рёбер e ≈ 10 + 10 + 11 + 9 + 8 + 6 = 54.

Z(G) = 54 - 42 + 2*1 = 12 + 2 = 14

ВЫВОД

В ходе работы проанализирована программа на C++, выполняющая генерацию двух случайных матриц 7×7, вывод диагоналей и вычисление их сумм. Построен граф потока управления. По формуле Мак-Кейба Z(G) = e - v + 2p цикломатическая сложность программы составила 14 (p = 1).

Значение превышает порог 10, что указывает на повышенную сложность и целесообразность рефакторинга (выделение общих операций в шаблонные функции снизит сложность отдельных модулей).

Метрика Мак-Кейба подтвердила свою актуальность: она наглядно показала минимальное количество тестовых прогонов (14), необходимых для полного покрытия всех ветвей и контуров программы, и позволила объективно оценить структурную сложность кода.