МИНОБРНАУКИ РОССИИ

Федеральное государственное бюджетное образовательное учреждение
высшего образования
«МИРЭА – Российский технологический университет»
РТУ МИРЭА

Институт Информационных технологий

Кафедра Математического обеспечения и стандартизации информационных технологий

ОТЧЕТ ПО ПРАКТИЧЕСКОЙ РАБОТЕ № 2
по дисциплине
«Структуры и алгоритмы обработки данных**»**

Тема: «Двумерный массив»

Выполнил студент группы ИКБО-68-23 Зуев Д.А.

Принял преподаватель Сартаков М.В.

Самостоятельная работа выполнена «__»_____202__г. *(подпись студента)*

«Зачтено» «__»_____202__г. *(подпись руководителя)*

Москва 2025

Практическая работа 2

1. Цель работы

Получение навыков по определению двумерного массива для структуры данных задачи и получение навыков по разработке алгоритмов операций на двумерном массиве в соответствии с задачей.

2. Постановка задачи

1. Разработать программу по обработке данных, представленных в задаче матрицей и реализованной в программе двумерным (многомерным) динамическим массивом.

a. Размеры массива должны определяться пользователем.

b. Двумерный массив определить как двойной указатель и выполнить его создание операцией new.

c. Разработать функции по реализации алгоритмов заполнения матрицы: с клавиатуры, датчиком случайных чисел. Разработать функции по реализации алгоритма вывода массива на экран построчно.

d. Выполнить декомпозицию задачи варианта, разработать алгоритм решения. Реализовать функцию, выполняющую задачу и отладить ее.

e. Разработать программу, демонстрирующую работу всех функций.

2. Разработать программу по задаче варианта с реализацией данных задачи с применением контейнера vector библиотеки STL.

a. Реализовать структуру хранения данных на основе шаблона <vector>, размеры определить при вводе с клавиатуры.

b. Разработать функции: заполнение структуры хранения исходных данных, вывода структуры хранения.

c. Выполнить декомпозицию задачи варианта, разработать алгоритм

решения. Реализовать функцию, выполняющую задачу и отладить ее.

d. Разработать программу, демонстрирующую работу всех функций.

Задание 1. Вариант 29. Дана квадратная матрица. Проверить, что в данной матрице произведение элементов, стоящих над побочной диагональю, равно произведению элементов, стоящих над главной диагональю.

Задание 2. Вариант 29. Дано множество из n (нечетное) точек на плоскости. Найти вертикальную медиану этого множества точек на плоскости в предположении, что никакие две точки не лежат на одной прямой. Медианой множества называется прямая, которая делит множество на два подмножества одинаковой мощности. Медиана строится через точку этого множества.

3. Решение

Задание 1

Задача разбивается на несколько этапов:

- Создание, заполнение и вывод матрицы.
- Проверка условия: вычисление произведений элементов над побочной и главной диагоналями.

Алгоритм проверки условия:

1. Инициализация переменных для хранения произведений элементов над главной и побочной диагоналями (productAboveMain, productAboveSecondary).
2. Пробег по элементам матрицы:
   - Если элемент находится над главной диагональю (индекс строки меньше индекса столбца), он умножается на productAboveMain.
   - Если элемент находится над побочной диагональю (сумма индексов строки и столбца меньше n - 1), он умножается на

productAboveSecondary.

3. Сравнение полученных произведений и возврат true, если произведения равны, иначе false.

```
// Функция для проверки условия задачи
bool checkMatrixCondition(int** matrix, int n) {
    int productAboveMain = 1;
    int productAboveSecondary = 1;

    for (int i = 0; i < n; ++i) {
        for (int j = 0; j < n; ++j) {
            if (i < j) {
                productAboveMain *= matrix[i][j];
            }
            if (i + j < n - 1) {
                productAboveSecondary *= matrix[i][j];
            }
        }
    }

    return productAboveMain == productAboveSecondary;
}
```

Рисунок 1 – Функция, реализующая задачу личного варианта

Функция createMatrix создает квадратную матрицу размером n x n, выделяя память под массив указателей, где каждый элемент является строкой. Она инициализирует строки, выделяя под них память, и возвращает указатель на матрицу, которая впоследствии может быть заполнена данными.

```
// Функция для создания и заполнения матрицы
int** createMatrix(int n) {
    int** matrix = new int* [n];
    for (int i = 0; i < n; ++i) {
        matrix[i] = new int[n];
    }
    return matrix;
}
```

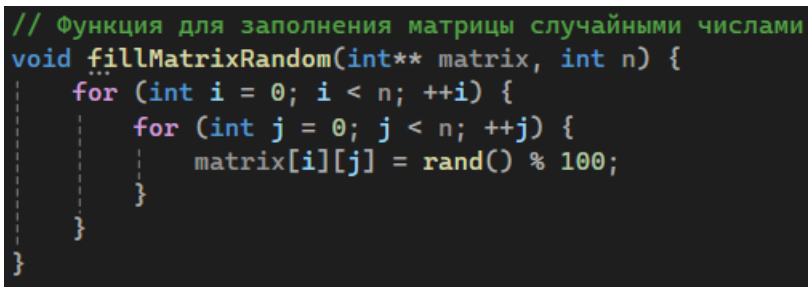
Рисунок 2 – Функция createMatrix

Функция fillMatrixFromKeyboard заполняет матрицу размером n x n, считывая значения с клавиатуры. Она использует два вложенных цикла для перебора элементов матрицы, запрашивая ввод каждого элемента и записывая его в соответствующую ячейку.

```
// Функция для заполнения матрицы с клавиатуры
void fillMatrixFromKeyboard(int** matrix, int n) {
    for (int i = 0; i < n; ++i) {
        for (int j = 0; j < n; ++j) {
            cin >> matrix[i][j];
        }
    }
}
```

Рисунок 3 – Функция fillMatrixFromKeyboard

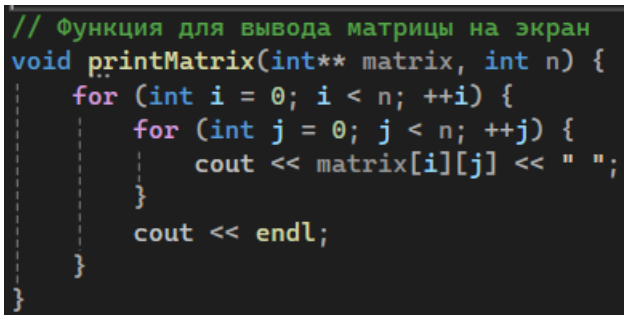
Функция fillMatrixRandom заполняет матрицу размером n x n случайными числами от 0 до 99. Используя два вложенных цикла, она перебирает элементы матрицы, присваивая каждому элементу случайное значение, что позволяет получить матрицу с произвольными числами.



```
// Функция для заполнения матрицы случайными числами
void fillMatrixRandom(int** matrix, int n) {
    for (int i = 0; i < n; ++i) {
        for (int j = 0; j < n; ++j) {
            matrix[i][j] = rand() % 100;
        }
    }
}
```

Рисунок 4 – Функция fillMatrixRandom

Функция printMatrix выводит на экран матрицу размером n x n. Используя два вложенных цикла, она проходит по каждому элементу матрицы, выводя его на экран с пробелом, чтобы представить в виде строк и столбцов, завершая строку символом новой строки.



```
// Функция для вывода матрицы на экран
void printMatrix(int** matrix, int n) {
    for (int i = 0; i < n; ++i) {
        for (int j = 0; j < n; ++j) {
            cout << matrix[i][j] << " ";
        }
        cout << endl;
    }
}
```

Рисунок 5 – Функция printMatrix

Задание 2

1. Ввод и хранение данных

- Описание: программа принимает от пользователя координаты точек и сохраняет их.

- Реализация:

- Структура Point представляет точку на плоскости и содержит два целочисленных поля x и y, которые обозначают координаты точки по горизонтали и вертикали соответственно.

```
struct Point {
    int x, y;
};
```

Рисунок 6 – Структура Point

- Функция fillPoints заполняет множество точек, которые сохраняются в векторе points. Она считывает количество точек n, а затем в цикле запрашивает ввод координат x и y для каждой точки с клавиатуры, добавляя заполненные точки в вектор с помощью метода push_back.

```
// Функция для заполнения множества точек с клавиатуры
void fillPoints(vector<Point>& points, int n) {
    for (int i = 0; i < n; ++i) {
        Point p;
        cin >> p.x >> p.y;
        points.push_back(p);
    }
}
```

Рисунок 6 – Функция fillPoints

2. Вывод данных

- Описание: вывод точек в формате (x, y) для визуализации.
- Реализация:
- Функция printPoints выводит на экран все точки из вектора points. Она перебирает каждую точку в векторе с помощью цикла for, и для каждой точки отображает её координаты x и y в виде пары, заключённой в скобки, с новой строки, что позволяет наглядно представить множество точек.

```
// Функция для вывода множества точек на экран
void printPoints(const vector<Point>& points) {
    for (const auto& p : points) {
        cout << "(" << p.x << ", " << p.y << ")" << endl;
    }
}
```

Рисунок 6 – Функция printPoints

3. Вычисление вертикальной медианы

- Описание: нахождение медианы x-координат.
- Реализация:

Функция findVerticalMedian находит вертикальную медиану множества точек, основываясь на их координатах x. Она извлекает все x-координаты из вектора points, сохраняет их в отдельном векторе x_coords, затем сортирует этот вектор и возвращает центральное значение, которое является медианой.

```
// Функция для нахождения вертикальной медианы множества точек
double findVerticalMedian(const vector<Point>& points) {
    vector<int> x_coords;
    for (const auto& p : points) {
        x_coords.push_back(p.x);
    }
    sort(x_coords.begin(), x_coords.end());
    return x_coords[x_coords.size() / 2];
}
```

Рисунок 7 – Функция findVerticalMedian

4. Тестирование

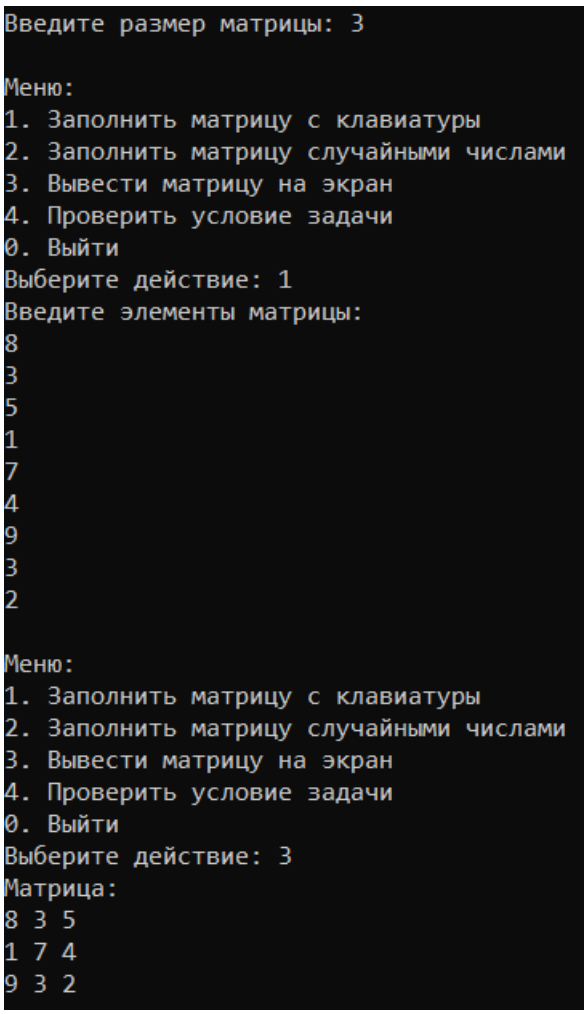


Рисунок 8 – Определение размера матрицы, ввод значений вручную и вывод матрицы

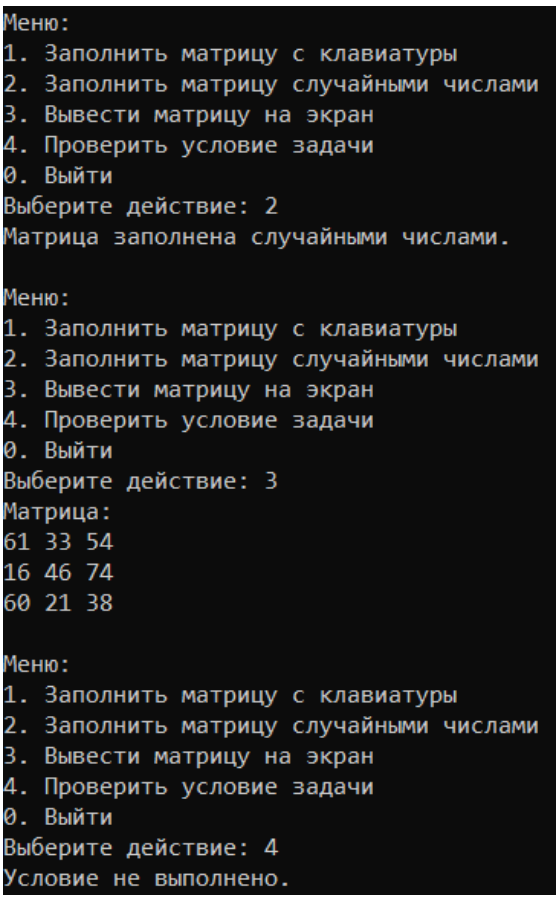


Рисунок 9 – Заполнение матрицы случайными числами и проверка условия (не выполнено)

```
Меню:
1. Заполнить матрицу с клавиатуры
2. Заполнить матрицу случайными числами
3. Вывести матрицу на экран
4. Проверить условие задачи
0. Выйти
Выберите действие: 1
Введите элементы матрицы:
3
3
3
3
3
3
3
3
3

Меню:
1. Заполнить матрицу с клавиатуры
2. Заполнить матрицу случайными числами
3. Вывести матрицу на экран
4. Проверить условие задачи
0. Выйти
Выберите действие: 3
Матрица:
3 3 3
3 3 3
3 3 3

Меню:
1. Заполнить матрицу с клавиатуры
2. Заполнить матрицу случайными числами
3. Вывести матрицу на экран
4. Проверить условие задачи
0. Выйти
Выберите действие: 4
Условие выполнено.
```

Рисунок 10 – Ручное заполнение матрицы и проверка условия (выполнено)

```
Введите количество точек: 5
Введите точки:
2
8
3
6
1
0
8
6
4
7
Точки:
(2, 8)
(3, 6)
(1, 0)
(8, 6)
(4, 7)
Вертикальная медиана: 3
```

Рисунок 11 – Тестирование задания 2

5. Вывод

В ходе выполнения задания была разработана программа для обработки данных, представленных в виде матрицы и множества точек на плоскости. Программа реализована на языке C++ с использованием динамических массивов и контейнера vector из библиотеки STL. Основные этапы разработки включали создание и заполнение матрицы, вывод матрицы на экран, проверку условия задачи для матрицы, а также создание и заполнение множества точек, вывод точек на экран и нахождение вертикальной медианы множества точек. Для удобства тестирования всех функций была добавлена возможность выбора действий через меню. Программа успешно прошла тестирование на различных входных данных, что подтверждает её корректность и работоспособность.

6. Исходный код

Исходный код задания 1:

```
#include <iostream>
#include <vector>
#include <cstdlib>
#include <ctime>

using namespace std;

// Функция для создания и заполнения матрицы
int** createMatrix(int n) {
    int** matrix = new int* [n];
    for (int i = 0; i < n; ++i) {
        matrix[i] = new int[n];
    }
    return matrix;
}

// Функция для заполнения матрицы с клавиатуры
void fillMatrixFromKeyboard(int** matrix, int n) {
    for (int i = 0; i < n; ++i) {
        for (int j = 0; j < n; ++j) {
            cin >> matrix[i][j];
        }
    }
}
```

```
// Функция для заполнения матрицы случайными числами
void fillMatrixRandom(int** matrix, int n) {
    for (int i = 0; i < n; ++i) {
        for (int j = 0; j < n; ++j) {
            matrix[i][j] = rand() % 100;
        }
    }
}

// Функция для вывода матрицы на экран
void printMatrix(int** matrix, int n) {
    for (int i = 0; i < n; ++i) {
        for (int j = 0; j < n; ++j) {
            cout << matrix[i][j] << " ";
        }
        cout << endl;
    }
}

// Функция для проверки условия задачи
bool checkMatrixCondition(int** matrix, int n) {
    int productAboveMain = 1;
    int productAboveSecondary = 1;

    for (int i = 0; i < n; ++i) {
        for (int j = 0; j < n; ++j) {
            if (i < j) {
                productAboveMain *= matrix[i][j];
            }
            if (i + j < n - 1) {
                productAboveSecondary *= matrix[i][j];
            }
        }
    }

    return productAboveMain == productAboveSecondary;
}

int main() {
    setlocale(0, "");
    srand(time(0));

    int n;
    cout << "Введите размер матрицы: ";
    cin >> n;

    int** matrix = createMatrix(n);

    int choice;
```

```
    do {
        cout << "\nМеню:" << endl;
        cout << "1. Заполнить матрицу с клавиатуры" << endl;
        cout << "2. Заполнить матрицу случайными числами" << endl;
        cout << "3. Вывести матрицу на экран" << endl;
        cout << "4. Проверить условие задачи" << endl;
        cout << "0. Выйти" << endl;
        cout << "Выберите действие: ";
        cin >> choice;

        switch (choice) {
        case 1:
            cout << "Введите элементы матрицы:" << endl;
            fillMatrixFromKeyboard(matrix, n);
            break;
        case 2:
            fillMatrixRandom(matrix, n);
            cout << "Матрица заполнена случайными числами." << endl;
            break;
        case 3:
            cout << "Матрица:" << endl;
            printMatrix(matrix, n);
            break;
        case 4:
            if (checkMatrixCondition(matrix, n)) {
                cout << "Условие выполнено." << endl;
            }
            else {
                cout << "Условие не выполнено." << endl;
            }
            break;
        case 0:
            cout << "Выход из программы." << endl;
            break;
        default:
            cout << "Неверный выбор. Попробуйте снова." << endl;
        }
    } while (choice != 0);

    // Освобождение памяти
    for (int i = 0; i < n; ++i) {
        delete[] matrix[i];
    }
    delete[] matrix;

    return 0;
}
```

Исходный код задания 2:

```
#include <iostream>
#include <vector>
#include <algorithm>

using namespace std;

struct Point {
    int x, y;
};

// Функция для заполнения множества точек с клавиатуры
void fillPoints(vector<Point>& points, int n) {
    for (int i = 0; i < n; ++i) {
        Point p;
        cin >> p.x >> p.y;
        points.push_back(p);
    }
}

// Функция для вывода множества точек на экран
void printPoints(const vector<Point>& points) {
    for (const auto& p : points) {
        cout << "(" << p.x << ", " << p.y << ")" << endl;
    }
}

// Функция для нахождения вертикальной медианы множества точек
double findVerticalMedian(const vector<Point>& points) {
    vector<int> x_coords;
    for (const auto& p : points) {
        x_coords.push_back(p.x);
    }
    sort(x_coords.begin(), x_coords.end());
    return x_coords[x_coords.size() / 2];
}

int main() {
    int n;
    cout << "Введите количество точек: ";
    cin >> n;

    vector<Point> points;
    cout << "Введите точки:" << endl;
    fillPoints(points, n);

    cout << "Точки:" << endl;
    printPoints(points);
```

```
    double median = findVerticalMedian(points);
    cout << "Вертикальная медиана: " << median << endl;

    return 0;
}
```