ЛАБОРАТОРНАЯ РАБОТА №5

ИССЛЕДОВАНИЕ АЛГОРИТМОВ ОБРАБОТКИ ДВУМЕРНЫХ МАССИВОВ С ПОМОЩЬЮ ФУНКЦИЙ

1. Цель работы:

Исследование основных принципов обработки двумерных массивов, исследование способов передачи параметров в функции. Приобретение практических навыков разработки алгоритмов обработки двумерных динамических массивов средствами языков C/C++.

2. Постановка задачи и вариант задания:

Вариант 5: Посчитать сумму элементов всех строк, не содержащих положительных элементов.

Соседями элемента a_{ij} в матрице A назовем элементы a_{kl}, для которых верно следующее: $i-1 \le k \le i+1$, $j-1 \le l \le j+1$. Элемент матрицы называется локальным минимумом, если он строго меньше всех имеющихся у него соседей. Подсчитать количество локальных минимумов заданной матрицы размером 5x5.

3. Краткие теоретические сведения

Для выполнения лабораторной работы необходимо изучить особенности объявления, описания и вызова двумерных массивов в языках C/C++.

ХОД РАБОТЫ

4. Структурная схема алгоритма

Структурная схема алгоритма, а также всех функций представлены на рисунках 5.1 – 5.5:

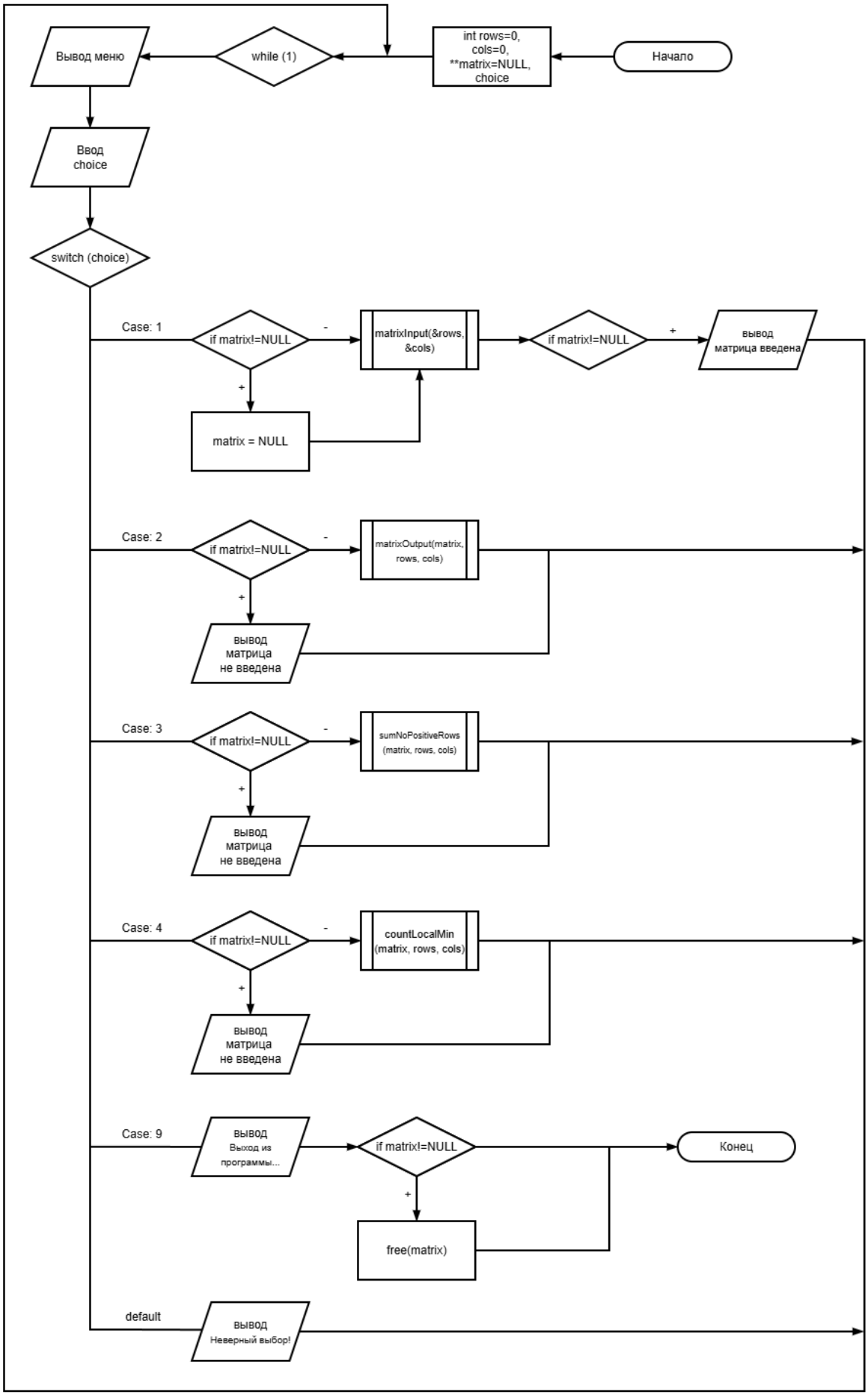

Рисунок 5.1 – Структурная схема функции main()

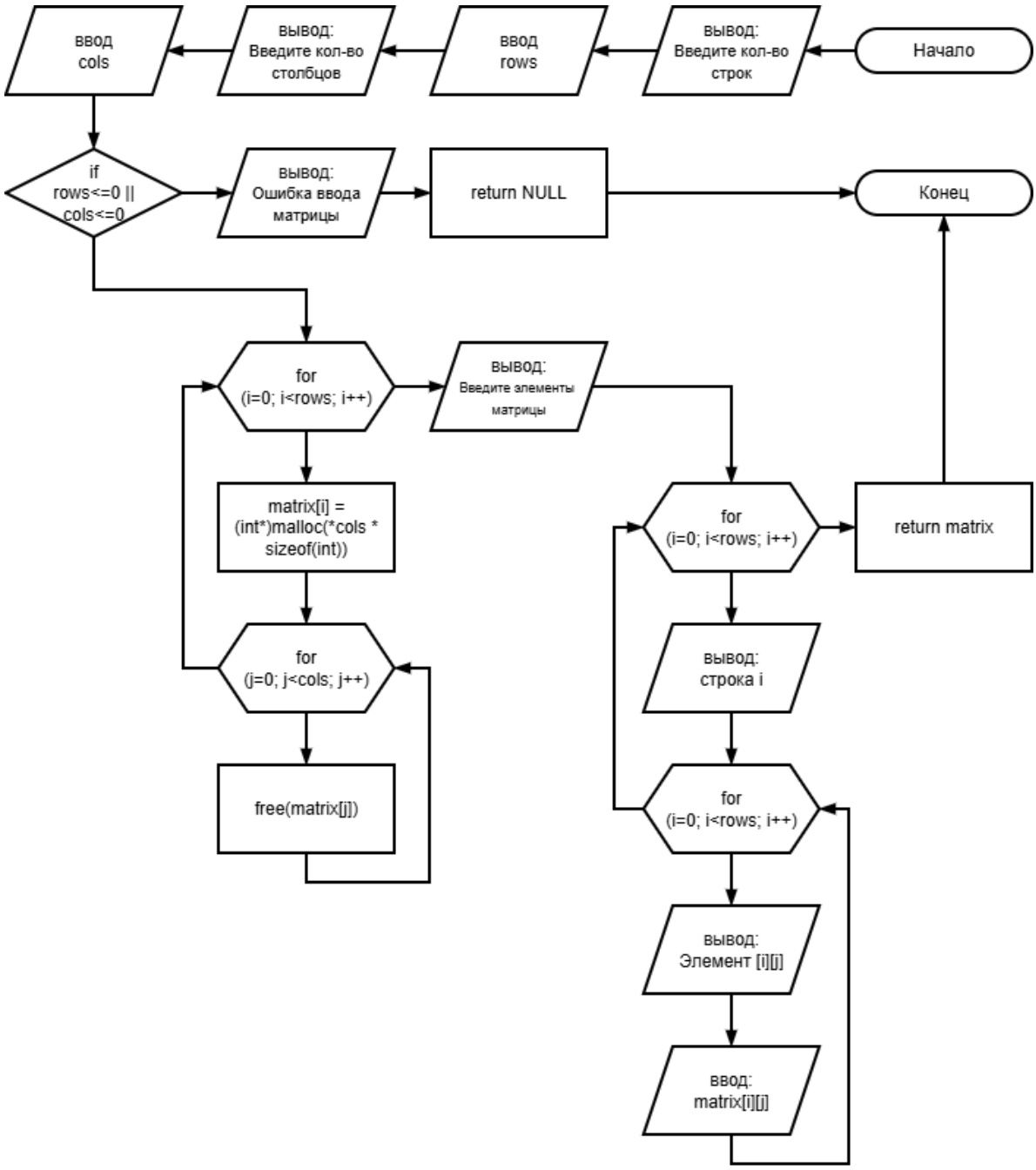

Рисунок 5.2 – Структурная схема функции matrixInput(*cols, *rows)

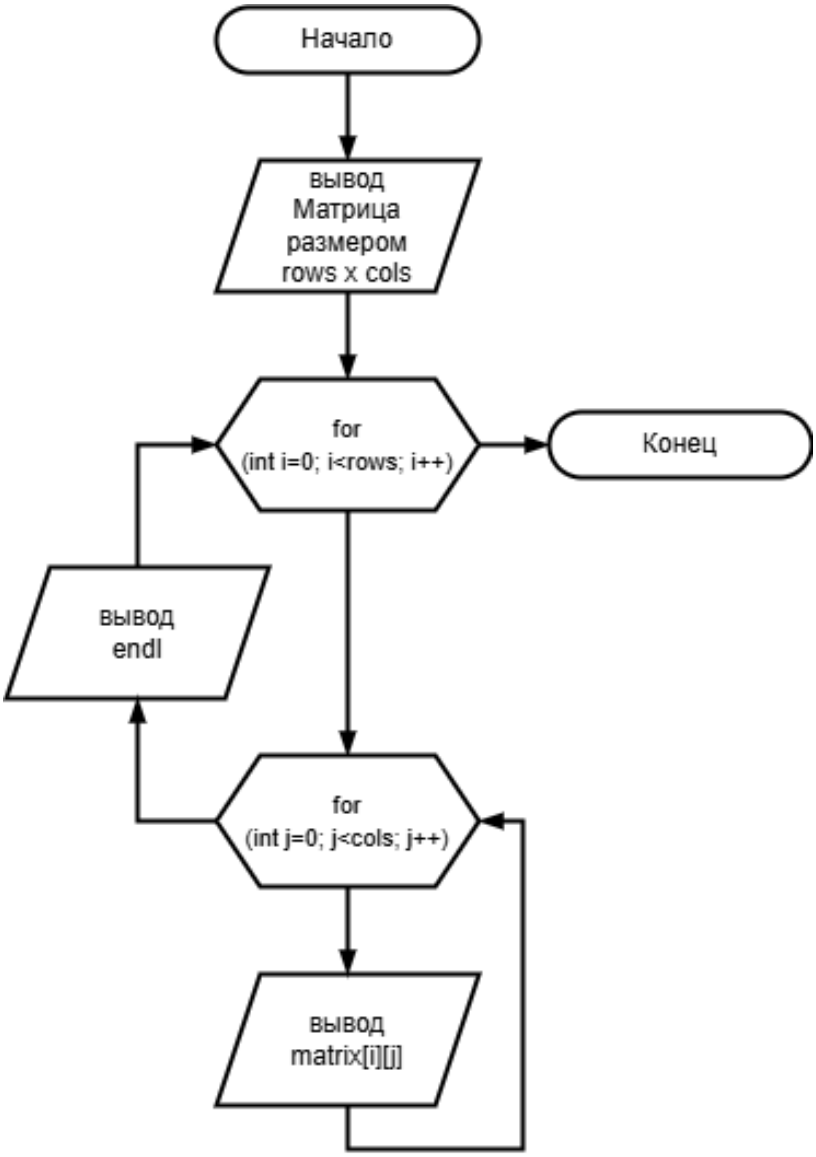

Рисунок 5.3 – Структурная схема функции matrixOutput(**matrix, rows, cols)

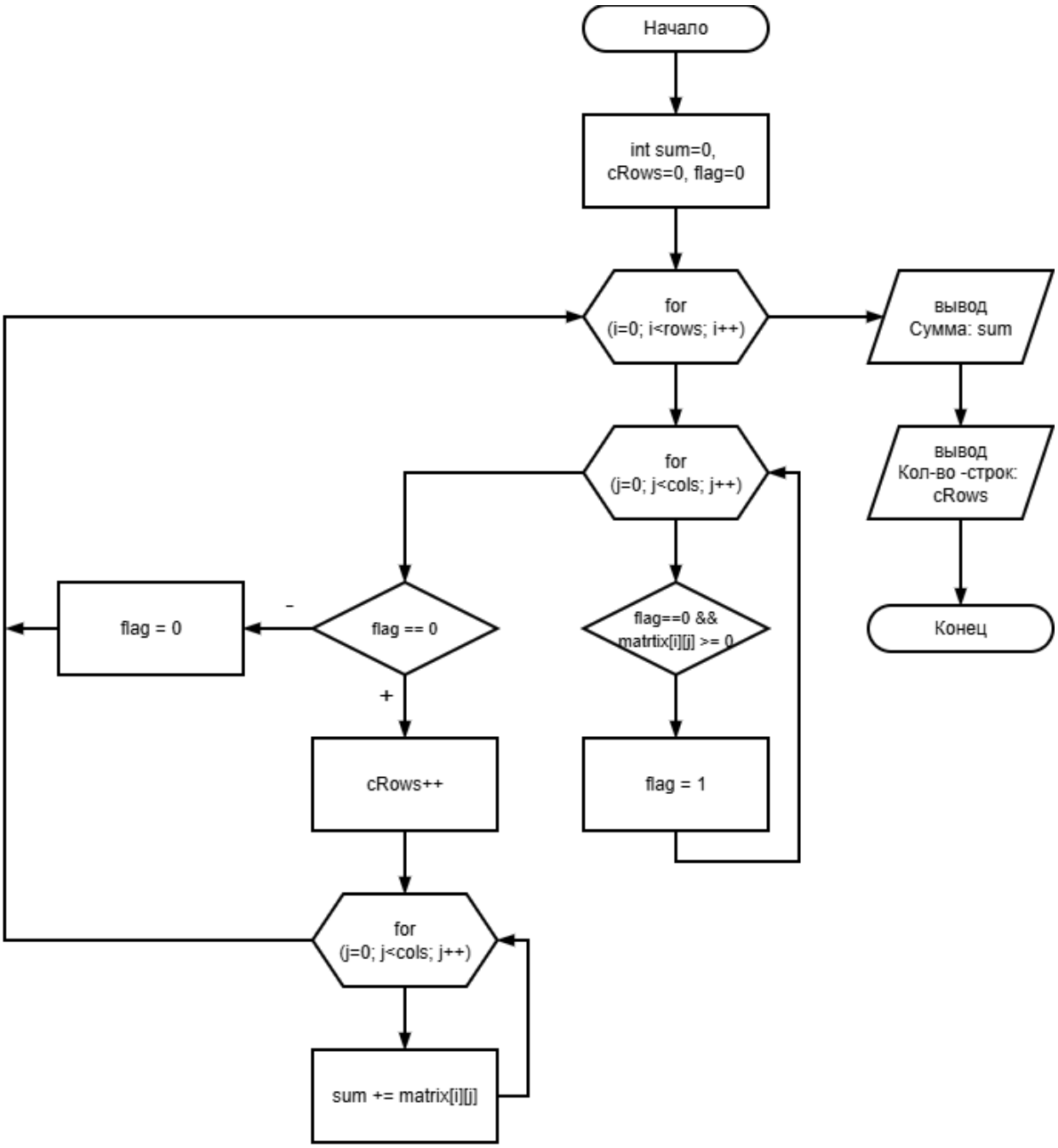

Рисунок 5.4 – Структурная схема функции sumNoPositiveRows(**matrix, cols, rows)

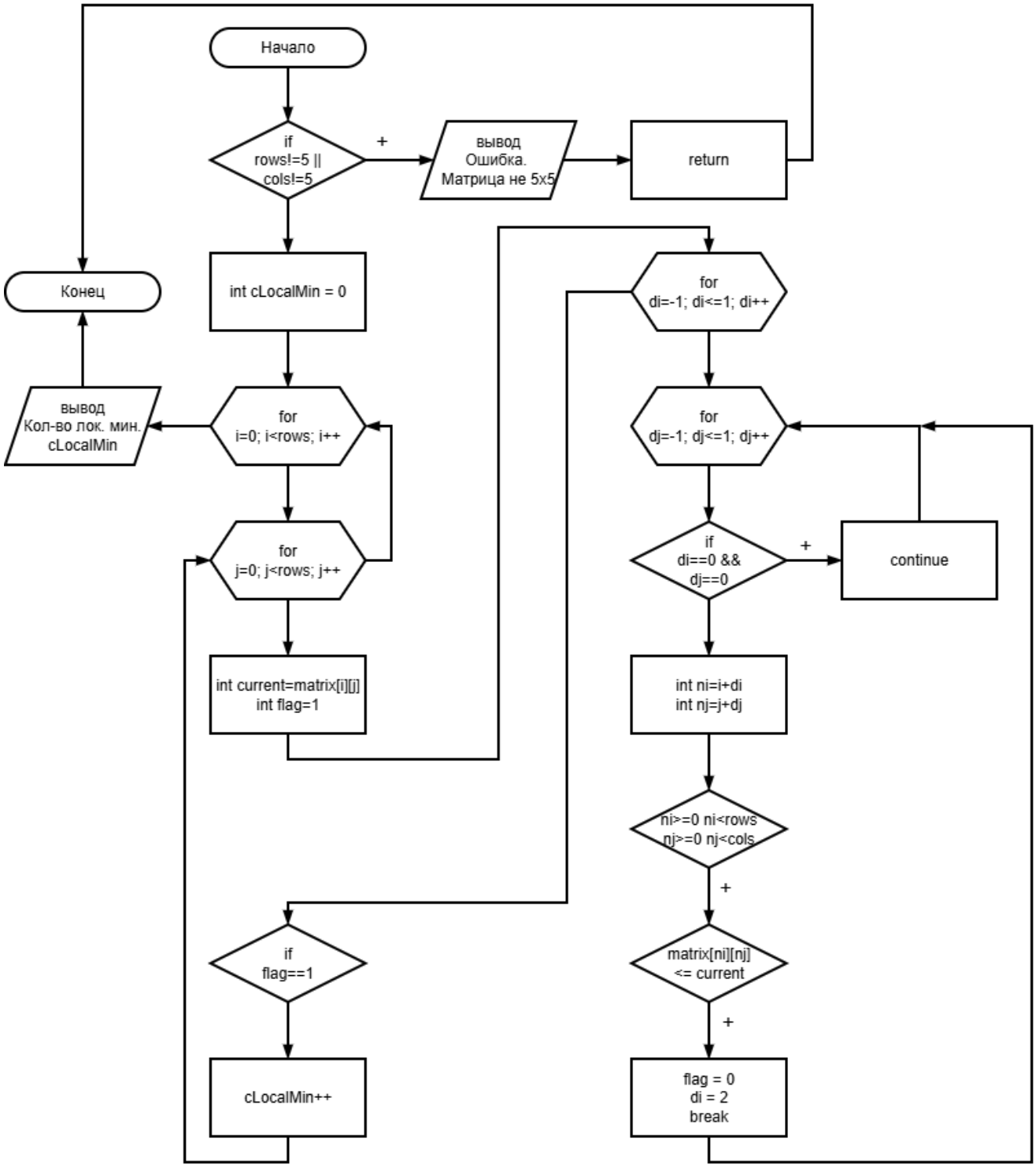

Рисунок 5.5 – Структурная схема функции countLocalMin(**matrix, cols, rows)

5. Текст кода на языке С. Тестирование и отладка программы

```
#include <stdio.h>
#include <math.h>
#include <stdlib.h>

// Объявление функций
int** matrixInput(int* rows, int* cols);
void matrixOutput(int** matrix, const int rows, const int cols);
void sumNoPositiveRows(int** matrix, const int rows, const int
cols);
void countLocalMin(int** matrix, const int rows, const int cols);

int main() {
    int **matrix = NULL;
    int rows = 0, cols = 0, choice;

    while (1) {
        printf("\n======= М Е Н Ю =======\n");
        printf("1. Ввести новую матрицу\n");
        printf("2. Вывести текущую матрицу\n");
        printf("3. Найти сумму полностью отрицательных строк\n");
        printf("4. Найти количество локальных минимумов (только
для матриц 5x5)\n");
        printf("9. Выход\n");
        printf("-> ");
        scanf("%d", &choice);
        printf("\n");

        switch (choice)
        {
        case 1:
            if (matrix != NULL) {
                free(matrix);
            }

            matrix = matrixInput(&rows, &cols);
            if (matrix != NULL) {
                printf("Матрица успешно введена.\n");
            }
            break;

        case 2:
            if (matrix == NULL) {
                printf("Матрица не введена!\n");
            }
            else {
                matrixOutput(matrix, rows, cols);
            }
            break;

        case 3:
            if (matrix == NULL) {
                printf("Матрица не введена!\n");
```

```
                }
                else {
                    sumNoPositiveRows(matrix, rows, cols);
                }
                break;

            case 4:
                if (matrix == NULL) {
                    printf("Матрица не введена!\n");
                }
                else {
                    countLocalMin(matrix, rows, cols);
                }
                break;

            case 9:
                printf("Выход из программы...\n");
                if (matrix != NULL) {
                    free(matrix);
                }
                return 0;
                break;

            default:
                printf("Неверный выбор! Попробуйте снова.\n");
            }
        }
    }

    int** matrixInput(int* rows, int* cols) {
        printf("Введите количество строк матрицы: ");
        scanf("%d", rows);
        printf("Введите количество столбцов матрицы: ");
        scanf("%d", cols);
        // Проверка корректности входных данных
        if (*rows <= 0 || *cols <= 0) {
            printf("Ошибка:    размеры    матрицы    должны    быть
положительными числами\n");
            return NULL;
        }

        // Выделение памяти под массив указателей на строки
        int** matrix = (int**)malloc(*rows * sizeof(int*));
        if (matrix == NULL) {
            printf("Ошибка выделения памяти для указателей строк\n");
            return NULL;
        }

        // Выделение памяти для каждой строки
        for (int i = 0; i < *rows; i++) {
            matrix[i] = (int*)malloc(*cols * sizeof(int));
            if (matrix[i] == NULL) {
                printf("Ошибка выделения памяти для строки %d\n", i);
```

```
                // Освобождаем уже выделенную память при ошибке
                for (int j = 0; j < i; j++) {
                    free(matrix[j]);
                }
                free(matrix);
                return NULL;
            }
        }

        // Ввод элементов матрицы с клавиатуры
        printf("Введите элементы матрицы %dx%d:\n", *rows, *cols);
        for (int i = 0; i < *rows; i++) {
            printf("Строка %d: \n", i + 1);
            for (int j = 0; j < *cols; j++) {
                printf("Элемент [%d][%d]: ", i+1, j+1);
                scanf("%d", &matrix[i][j]);
            }
        }

        return matrix;
    }

    void matrixOutput(int** matrix, const int rows, const int cols)
{
        printf("Введённая матрица размером %dx%d:\n", rows, cols);
        for (int i = 0; i < rows; i++) {
            for (int j = 0; j < cols; j++) {
                printf("%5d ", matrix[i][j]);
            }
            printf("\n"); // Переход на следущую строку
        }
    }

    void sumNoPositiveRows(int** matrix, const int rows, const int
cols) {
        int sum = 0, cRows = 0, flag = 0;
        for (int i = 0; i < rows; i++) {
            for (int j = 0; j < cols; j++) {
                if (flag == 0 && matrix[i][j] >= 0) {
                    flag = 1;
                }
            }
            if (flag == 0) {
                cRows++;  //  Добавляем  +1  к  количеству  полностью
отрицательных строк
                for (int j = 0; j < cols; j++) {
                    sum += matrix[i][j];
                }
            }
            else {
                flag = 0; // Обнуление флага при переходе на следующую
строку
```

```
            }
        }
        printf("Сумма полностью отрицательных строк: %d\n", sum);
        printf("Количество полностью отрицательных строк: %d\n",
cRows);
    }

    void countLocalMin(int** matrix, const int rows, const int cols)
{
        // Проверка размера матрицы
        if (rows != 5 && cols != 5) {
            printf("Введённая матрица имеет размер %dx%d. Для
выполнения данной функции необходима матрица 5x5!\n", rows, cols);
            return;
        }

        int cLocalMin = 0;
        for (int i = 0; i < rows; i++) {
            for (int j = 0; j < cols; j++) {
                int current = matrix[i][j];
                int flag = 1;

                // Проверяем всех соседей
                for (int di = -1; di <= 1; di++) {
                    for (int dj = -1; dj <= 1; dj++) {
                        if (di == 0 && dj == 0) {
                            continue;
                        }

                        int ni = i + di;
                        int nj = j + dj;

                        // Проверяем границы
                        if (ni >= 0 && ni < rows && nj >= 0 && nj <
cols) {
                            if (matrix[ni][nj] <= current) {
                                flag = 0;
                                //  Прерываем  оба  цикла  проверки
соседей
                                di = 2; // Выход из внешнего цикла
                                break;
                            }
                        }
                    }
                }
                if (flag == 1) {
                    cLocalMin++;
                }
            }
        }

        printf("Количество локальных минимумов: %d\n", cLocalMin);
    }
```

На Рисунках 5.6, 5.7 отображены результаты работы кода согласно всем условиям по заданию для языка C.

```
======= М Е Н Ю =======
1. Ввести новую матрицу
2. Вывести текущую матрицу
3. Найти сумму полностью отрицательных строк
4. Найти количество локальных минимумов (только для матриц 5x5)
9. Выход
-> 1

Введите количество строк матрицы: 2
Введите количество столбцов матрицы: 4
Введите элементы матрицы 2x4:
Строка 1:
Элемент [1][1]: 1
Элемент [1][2]: 2
Элемент [1][3]: 5
Элемент [1][4]: 2
Строка 2:
Элемент [2][1]: -1
Элемент [2][2]: -4
Элемент [2][3]: -2
Элемент [2][4]: -5
Матрица успешно введена.

======= М Е Н Ю =======
1. Ввести новую матрицу
2. Вывести текущую матрицу
3. Найти сумму полностью отрицательных строк
4. Найти количество локальных минимумов (только для матриц 5x5)
9. Выход
-> 2

Введённая матрица размером 2x4:
    1     2     5     2
   -1    -4    -2    -5

======= М Е Н Ю =======
1. Ввести новую матрицу
2. Вывести текущую матрицу
3. Найти сумму полностью отрицательных строк
4. Найти количество локальных минимумов (только для матриц 5x5)
9. Выход
-> 3

Сумма полностью отрицательных строк: -12
Количество полностью отрицательных строк: 1

======= М Е Н Ю =======
1. Ввести новую матрицу
2. Вывести текущую матрицу
3. Найти сумму полностью отрицательных строк
4. Найти количество локальных минимумов (только для матриц 5x5)
9. Выход
-> 4

Введённая матрица имеет размер 2x4. Для выполнения данной функции необходима матрица 5x5!
```

Рисунок 5.6 – Результат работы программы на языке C

```
======= М Е Н Ю =======
1. Ввести новую матрицу
2. Вывести текущую матрицу
3. Найти сумму полностью отрицательных строк
4. Найти количество локальных минимумов (только для матриц 5x5)
9. Выход
-> 2

Введённая матрица размером 5x5:
   24     3    12    32    54
   -3    -4    -6    -2    -4
    1    65    -2   -32    77
   -1    -1    -1    -1    -1
   23    -3    52    66    -4

======= М Е Н Ю =======
1. Ввести новую матрицу
2. Вывести текущую матрицу
3. Найти сумму полностью отрицательных строк
4. Найти количество локальных минимумов (только для матриц 5x5)
9. Выход
-> 3

Сумма полностью отрицательных строк: -24
Количество полностью отрицательных строк: 2

======= М Е Н Ю =======
1. Ввести новую матрицу
2. Вывести текущую матрицу
3. Найти сумму полностью отрицательных строк
4. Найти количество локальных минимумов (только для матриц 5x5)
9. Выход
-> 4

Количество локальных минимумов: 3

======= М Е Н Ю =======
1. Ввести новую матрицу
2. Вывести текущую матрицу
3. Найти сумму полностью отрицательных строк
4. Найти количество локальных минимумов (только для матриц 5x5)
9. Выход
-> 5

Неверный выбор! Попробуйте снова.

======= М Е Н Ю =======
1. Ввести новую матрицу
2. Вывести текущую матрицу
3. Найти сумму полностью отрицательных строк
4. Найти количество локальных минимумов (только для матриц 5x5)
9. Выход
-> 9

Выход из программы...
```

Рисунок 5.7 – Результат работы программы на языке С

6. Текст кода на языке C++. Тестирование и отладка программы

```
#include <iostream>
#include <math.h>
#include <stdlib.h>
#include <iomanip>

using namespace std;

// Объявление функций
int **matrixInput(int *rows, int *cols);
void matrixOutput(int **matrix, const int rows, const int cols);
void sumNoPositiveRows(int **matrix, const int rows, const int
cols);
void countLocalMin(int **matrix, const int rows, const int cols);

int main()
{
    int **matrix = NULL;
    int rows = 0, cols = 0, choice;

    while (1) {
        cout << endl << "======= М Е Н Ю =======" << endl;
        cout << "1. Ввести новую матрицу" << endl;
        cout << "2. Вывести текущую матрицу" << endl;
        cout << "3. Найти сумму полностью отрицательных строк" <<
endl;
        cout << "4. Найти количество локальных минимумов (только
для матриц 5x5)" << endl;
        cout << "9. Выход" << endl;
        cout << "-> ";
        cin >> choice;

        switch (choice)
        {
        case 1:
            if (matrix != NULL)
            {
                free(matrix);
            }

            matrix = matrixInput(&rows, &cols);
            if (matrix != NULL)
            {
                cout << "Матрица успешно введена." << endl;
            }
            break;

        case 2:
            if (matrix == NULL)
            {
                cout << "Матрица не введена!" << endl;
            }
            else
```

```
            {
                matrixOutput(matrix, rows, cols);
            }
            break;

        case 3:
            if (matrix == NULL)
            {
                cout << "Матрица не введена!" << endl;
            }
            else
            {
                sumNoPositiveRows(matrix, rows, cols);
            }
            break;

        case 4:
            if (matrix == NULL)
            {
                cout << "Матрица не введена!" << endl;
            }
            else
            {
                countLocalMin(matrix, rows, cols);
            }
            break;

        case 9:
            cout << "Выход из программы..." << endl;
            if (matrix != NULL)
            {
                free(matrix);
            }
            return 0;
            break;

        default:
            cout << endl
                 << "Неверный выбор! Попробуйте снова." << endl;
        }
    }
}

int **matrixInput(int *rows, int *cols)
{

    cout << endl
         << "Введите количество строк матрицы: ";
    cin >> *rows;
    cout << "Введите количество столбцов матрицы: ";
    cin >> *cols;

    // Проверка корректности входных данных
```

```
        if (*rows <= 0 || *cols <= 0)
        {
            cout << "Ошибка: размеры матрицы должны быть
положительными числами" << endl;
            return NULL;
        }

        // Выделение памяти под массив указателей на строки
        int **matrix = (int **)malloc(*rows * sizeof(int *));
        if (matrix == NULL)
        {
            cout << "Ошибка выделения памяти для указателей строк" <<
endl;
            return NULL;
        }

        // Выделение памяти для каждой строки
        for (int i = 0; i < *rows; i++)
        {
            matrix[i] = (int *)malloc(*cols * sizeof(int));
            if (matrix[i] == NULL)
            {
                cout << "Ошибка выделения памяти для строки " << i <<
"." << endl;

                // Освобождаем уже выделенную память при ошибке
                for (int j = 0; j < i; j++)
                {
                    free(matrix[j]);
                }
                free(matrix);
                return NULL;
            }
        }

        // Ввод элементов матрицы с клавиатуры
        cout << "Введите элементы матрицы " << *rows << "x" << *cols
<< ":" << endl;
        for (int i = 0; i < *rows; i++)
        {
            cout << "Строка " << i + 1 << ":" << endl;
            for (int j = 0; j < *cols; j++)
            {
                cout << "Элемент [" << i + 1 << "][" << j + 1 << "]:
";
                cin >> matrix[i][j];
            }
        }
        return matrix;
    }

    void matrixOutput(int **matrix, const int rows, const int cols)
    {
```

```
        cout << endl
             << "Введённая матрица размером " << rows << "x" << cols
<< ":" << endl;
        for (int i = 0; i < rows; i++)
        {
            for (int j = 0; j < cols; j++)
            {
                cout << setw(5) << matrix[i][j] << " ";
            }
            cout << endl; // Переход на следущую строку
        }
    }

    void sumNoPositiveRows(int **matrix, const int rows, const int
cols)
    {
        int sum = 0, cRows = 0, flag = 0;
        for (int i = 0; i < rows; i++)
        {
            for (int j = 0; j < cols; j++)
            {
                if (flag == 0 && matrix[i][j] >= 0)
                {
                    flag = 1;
                }
            }
            if (flag == 0)
            {
                cRows++; // Добавляем +1 к количеству полностью
отрицательных строк
                for (int j = 0; j < cols; j++)
                {
                    sum += matrix[i][j];
                }
            }
            else
            {
                flag = 0; // Обнуление флага при переходе на следующую
строку
            }
        }
        cout << endl
             << "Сумма полностью отрицательных строк: " << sum <<
endl;
        cout << "Количество полностью отрицательных строк: " << cRows
<< endl;
    }

    void countLocalMin(int **matrix, const int rows, const int cols)
    {
        // Проверка размера матрицы
        if (rows != 5 && cols != 5)
        {
```

```
            cout << endl << "Введённая матрица имеет размер " << rows
<< "x" << cols << ". Для выполнения данной функции необходима матрица
5x5!" << endl;
            return;
        }

        int cLocalMin = 0;

        for (int i = 0; i < rows; i++)
        {
            for (int j = 0; j < cols; j++)
            {
                int current = matrix[i][j];
                int flag = 1;

                // Проверяем всех соседей
                for (int di = -1; di <= 1; di++)
                {
                    for (int dj = -1; dj <= 1; dj++)
                    {
                        if (di == 0 && dj == 0)
                        {
                            continue;
                        }

                        int ni = i + di;
                        int nj = j + dj;

                        // Проверяем границы
                        if (ni >= 0 && ni < rows && nj >= 0 && nj <
cols)
                        {
                            if (matrix[ni][nj] <= current)
                            {
                                flag = 0;
                                //  Прерываем  оба  цикла  проверки
соседей
                                di = 2; // Выход из внешнего цикла
                                break;
                            }
                        }
                    }
                }
                if (flag == 1)
                {
                    cLocalMin++;
                }
            }
        }

        cout  <<  endl  <<  "Количество  локальных  минимумов:  "  <<
cLocalMin << endl;
    }
```

На Рисунках 5.8, 5.9 отображены результаты работы кода согласно всем условиям по заданию для языка C++.

```
======= М Е Н Ю =======
1. Ввести новую матрицу
2. Вывести текущую матрицу
3. Найти сумму полностью отрицательных строк
4. Найти количество локальных минимумов (только для матриц 5x5)
9. Выход
-> 1

Введите количество строк матрицы: 2
Введите количество столбцов матрицы: 3
Введите элементы матрицы 2x3:
Строка 1:
Элемент [1][1]: -4
Элемент [1][2]: -6
Элемент [1][3]: -17
Строка 2:
Элемент [2][1]: 2
Элемент [2][2]: -4
Элемент [2][3]: 1
Матрица успешно введена.

======= М Е Н Ю =======
1. Ввести новую матрицу
2. Вывести текущую матрицу
3. Найти сумму полностью отрицательных строк
4. Найти количество локальных минимумов (только для матриц 5x5)
9. Выход
-> 2

Введённая матрица размером 2x3:
   -4    -6   -17
    2    -4     1

======= М Е Н Ю =======
1. Ввести новую матрицу
2. Вывести текущую матрицу
3. Найти сумму полностью отрицательных строк
4. Найти количество локальных минимумов (только для матриц 5x5)
9. Выход
-> 3

Сумма полностью отрицательных строк: -27
Количество полностью отрицательных строк: 1

======= М Е Н Ю =======
1. Ввести новую матрицу
2. Вывести текущую матрицу
3. Найти сумму полностью отрицательных строк
4. Найти количество локальных минимумов (только для матриц 5x5)
9. Выход
-> 45

Неверный выбор! Попробуйте снова.
```

Рисунок 5.8 – Результат работы программы на языке C++

```
======= М Е Н Ю =======
1. Ввести новую матрицу
2. Вывести текущую матрицу
3. Найти сумму полностью отрицательных строк
4. Найти количество локальных минимумов (только для матриц 5x5)
9. Выход
-> 2

Введённая матрица размером 5x5:
   45    65     2   543     2
   -4    -3    45    34    12
   32    -5    -3    12    43
   34    54    77    89    -2
   34    -6    76    77    32

======= М Е Н Ю =======
1. Ввести новую матрицу
2. Вывести текущую матрицу
3. Найти сумму полностью отрицательных строк
4. Найти количество локальных минимумов (только для матриц 5x5)
9. Выход
-> 4

Количество локальных минимумов: 4

======= М Е Н Ю =======
1. Ввести новую матрицу
2. Вывести текущую матрицу
3. Найти сумму полностью отрицательных строк
4. Найти количество локальных минимумов (только для матриц 5x5)
9. Выход
-> 9
Выход из программы..._
```

Рисунок 5.9 – Результат работы программы на языке C++

ВЫВОД

В ходе лабораторной работы были исследованы особенности представления и обработки двумерных динамических массивов в языках C/C++. На практике были успешно реализованы алгоритмы работы с двумерными массивами, включая их динамическое создание, обработку и освобождение памяти. Экспериментально подтверждена и изучена неразрывная связь между указателями и двумерными массивами, лежащая в основе управления памятью. Приобретены навыки эффективного использования динамического выделения памяти под двумерные массивы на языках C/C++.