ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ  
«РОССИЙСКИЙ УНИВЕРСИТЕТ ТРАНСПОРТА»  
(РУТ (МИИТ))

Институт транспортной техники и систем управления

Кафедра «Управление и защита информации»

ОТЧЁТ  
О ПРАКТИЧЕСКОЙ РАБОТЕ № 1

По дисциплине «Языки программирования»

Выполнил: ст. гр. ТКИ – 141

Самусенкова В.Н

Проверил: к.т.н., доц.

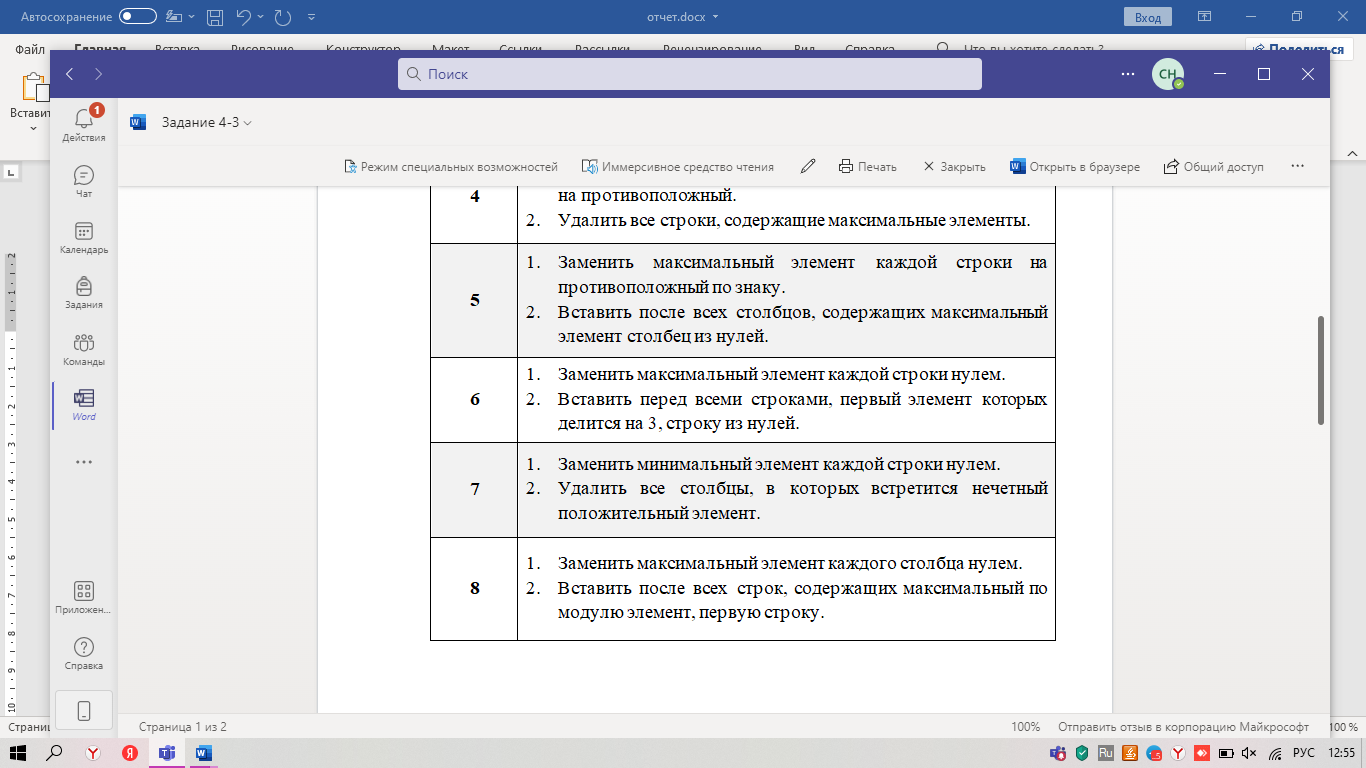
Васильева М. А.

Москва 2022

**Задание 4-3**

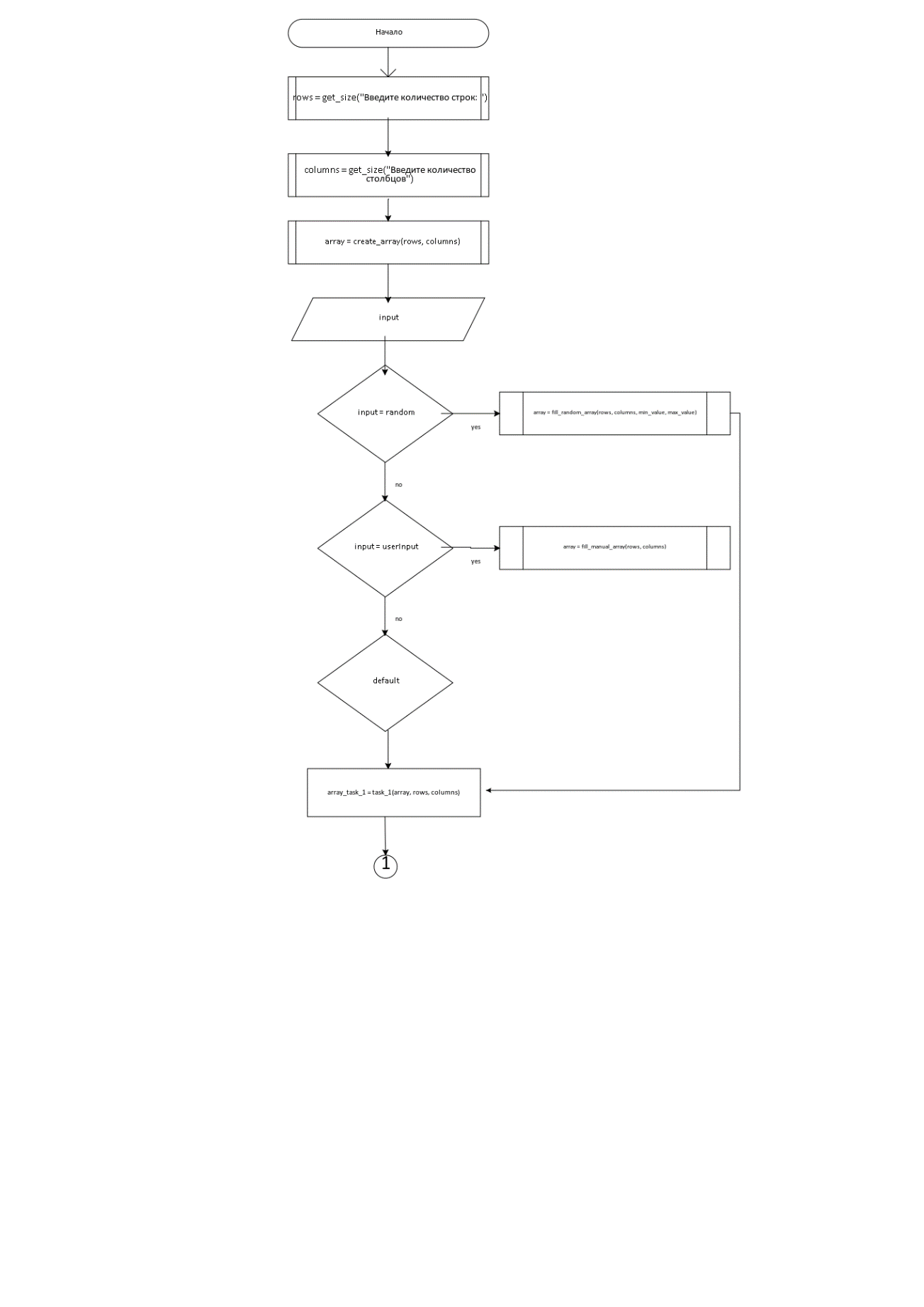
**1. Формулировка задачи**

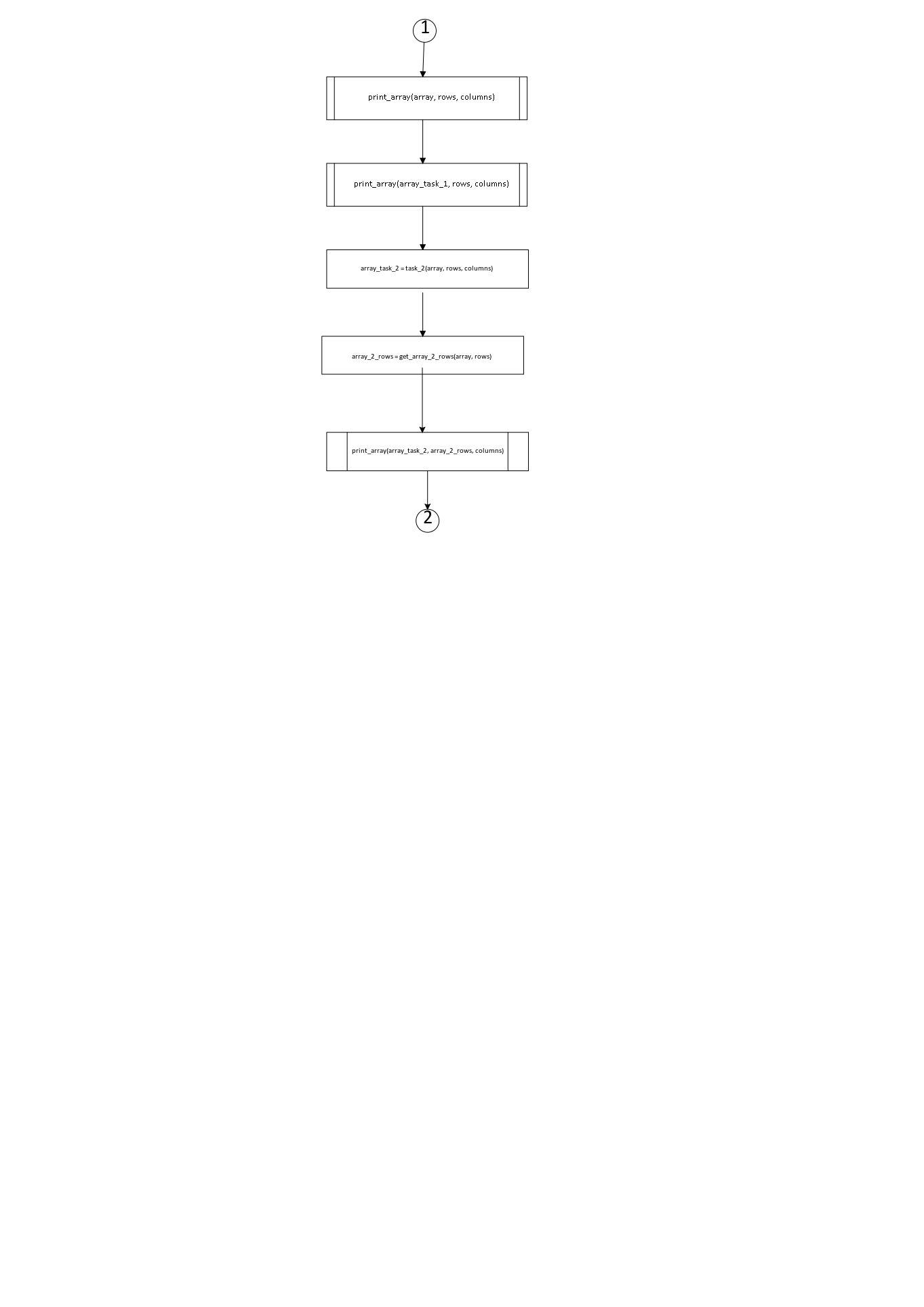
Создать многомерный массив nˣm из n целых чисел. Реализовать возможность заполнения массива, как случайными числами, так и с помощью клавиатуры по желанию пользователя. Пункты задания организовать в виде функций (методов). Вывести массив на экран.

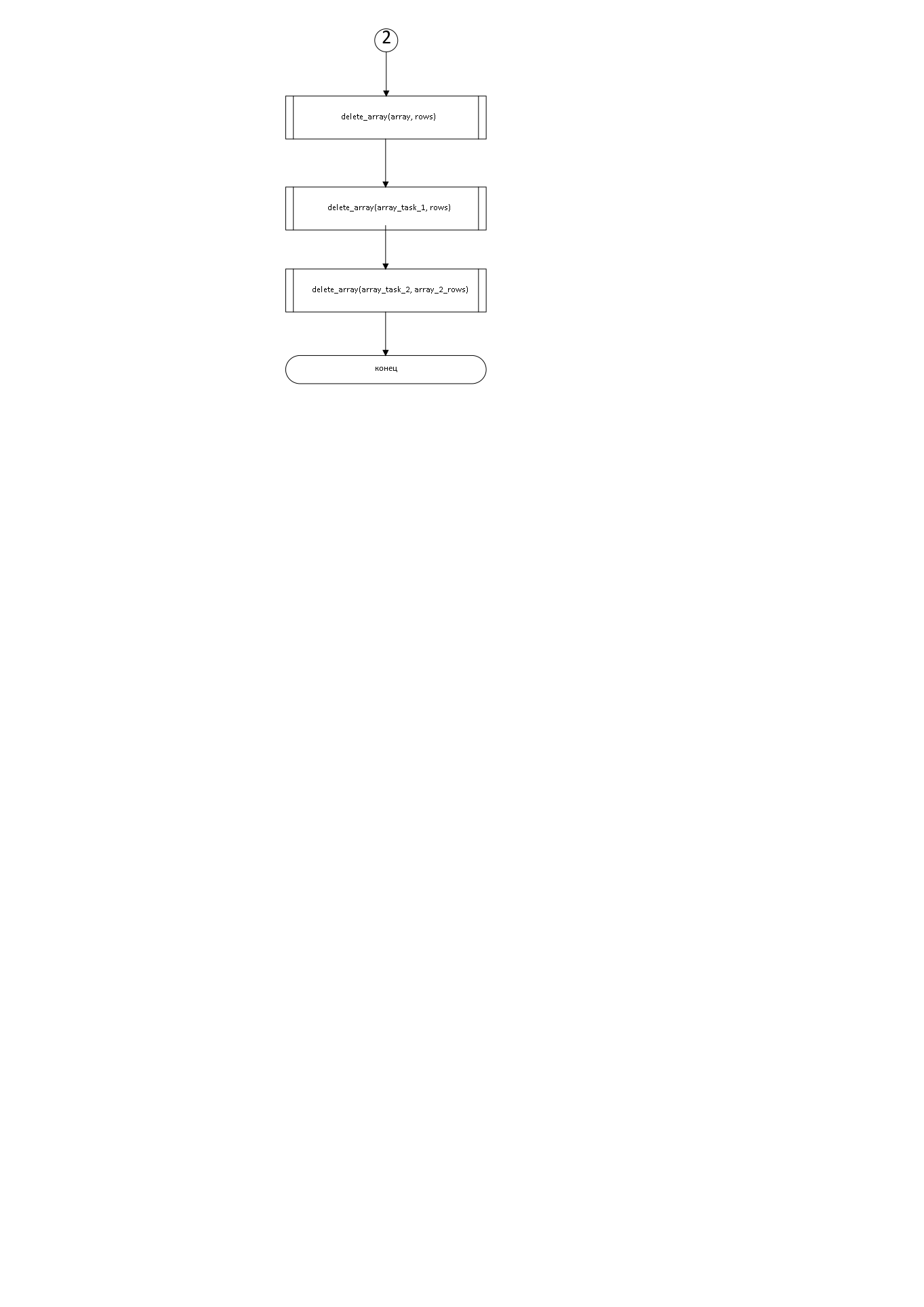


**2. Блок-схемы алгоритмов функций представлены на рисунках**

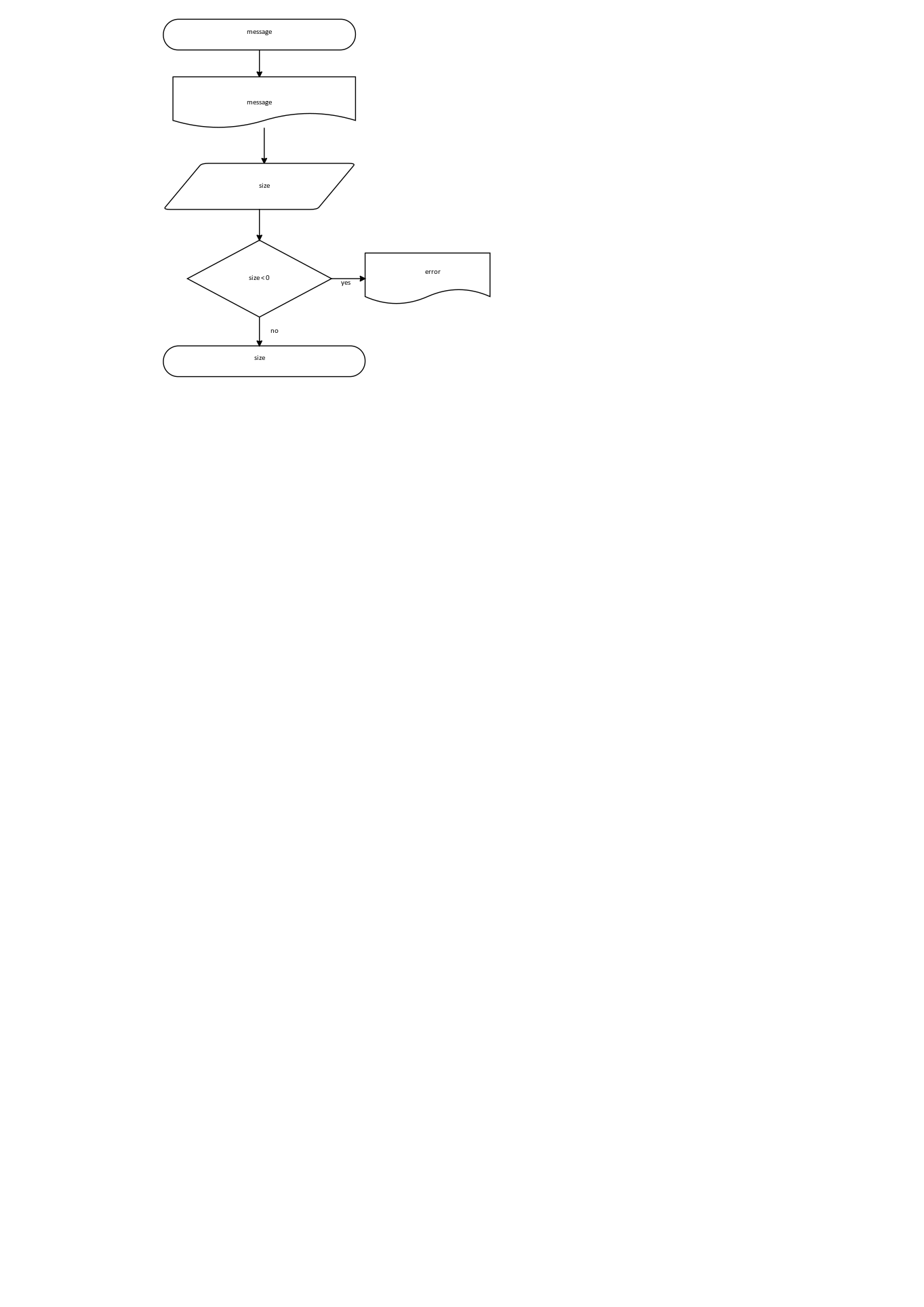
**(Рисунок 1-11)**



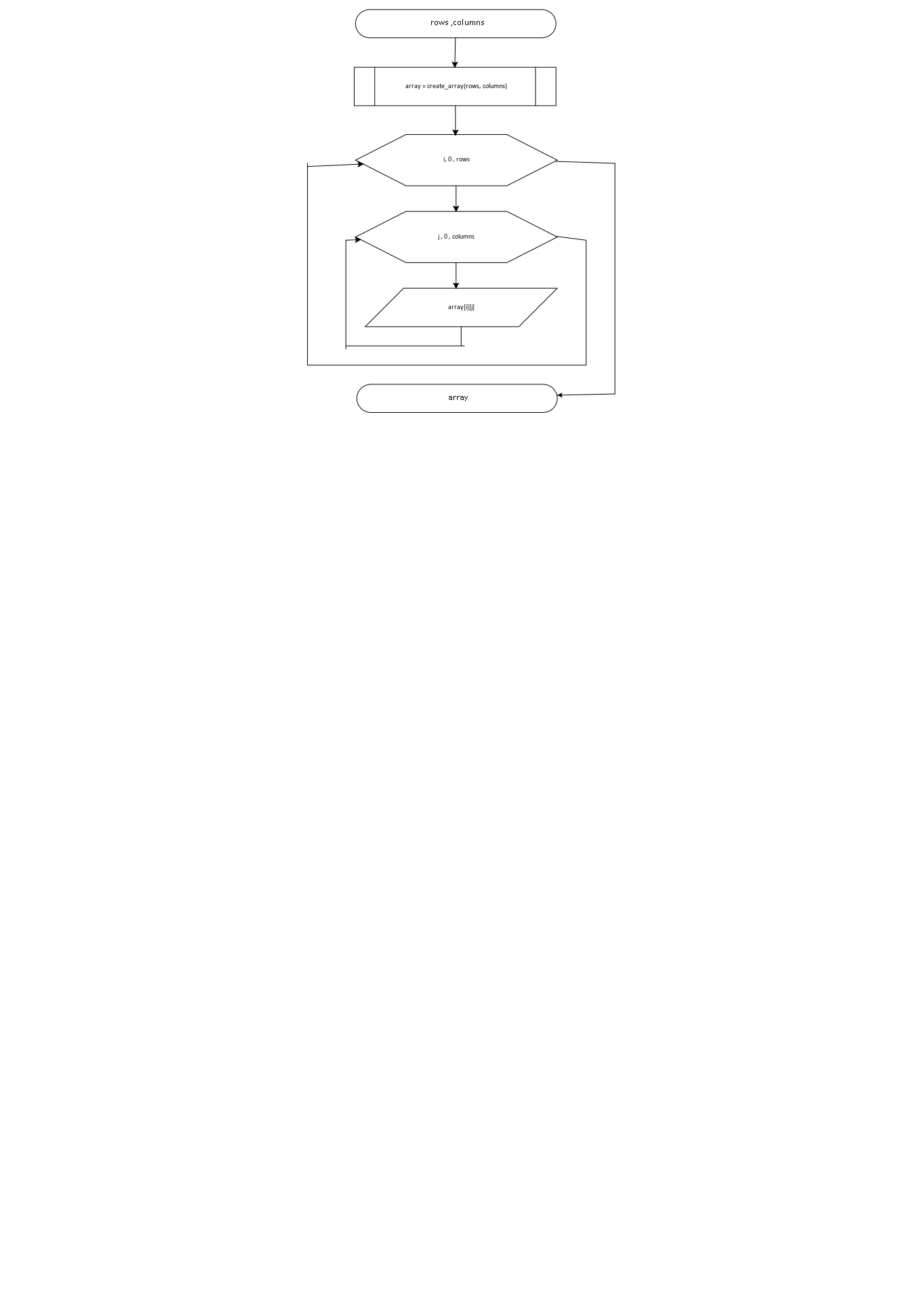




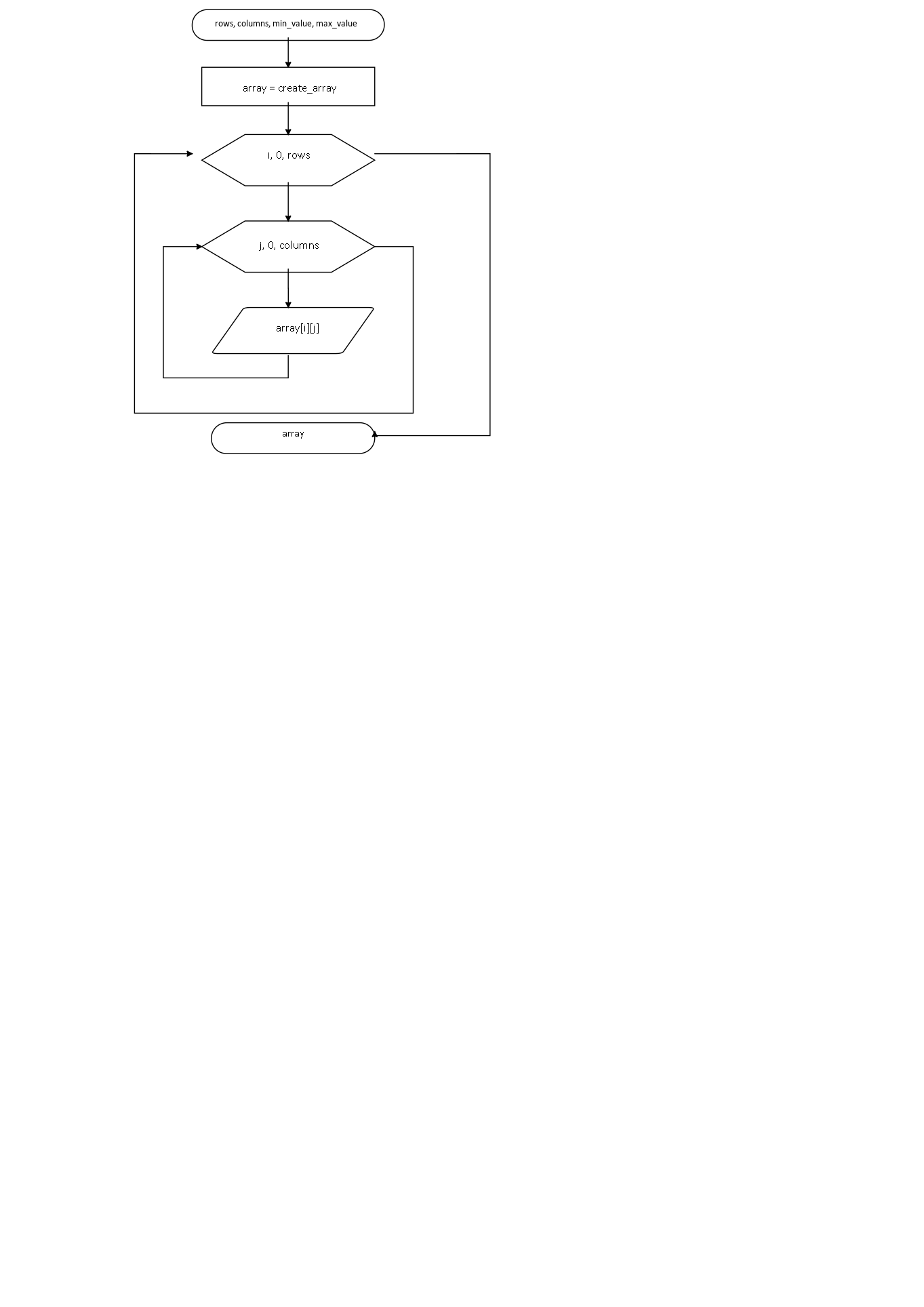
**Рисунок 1 Блок-Схема функции main**



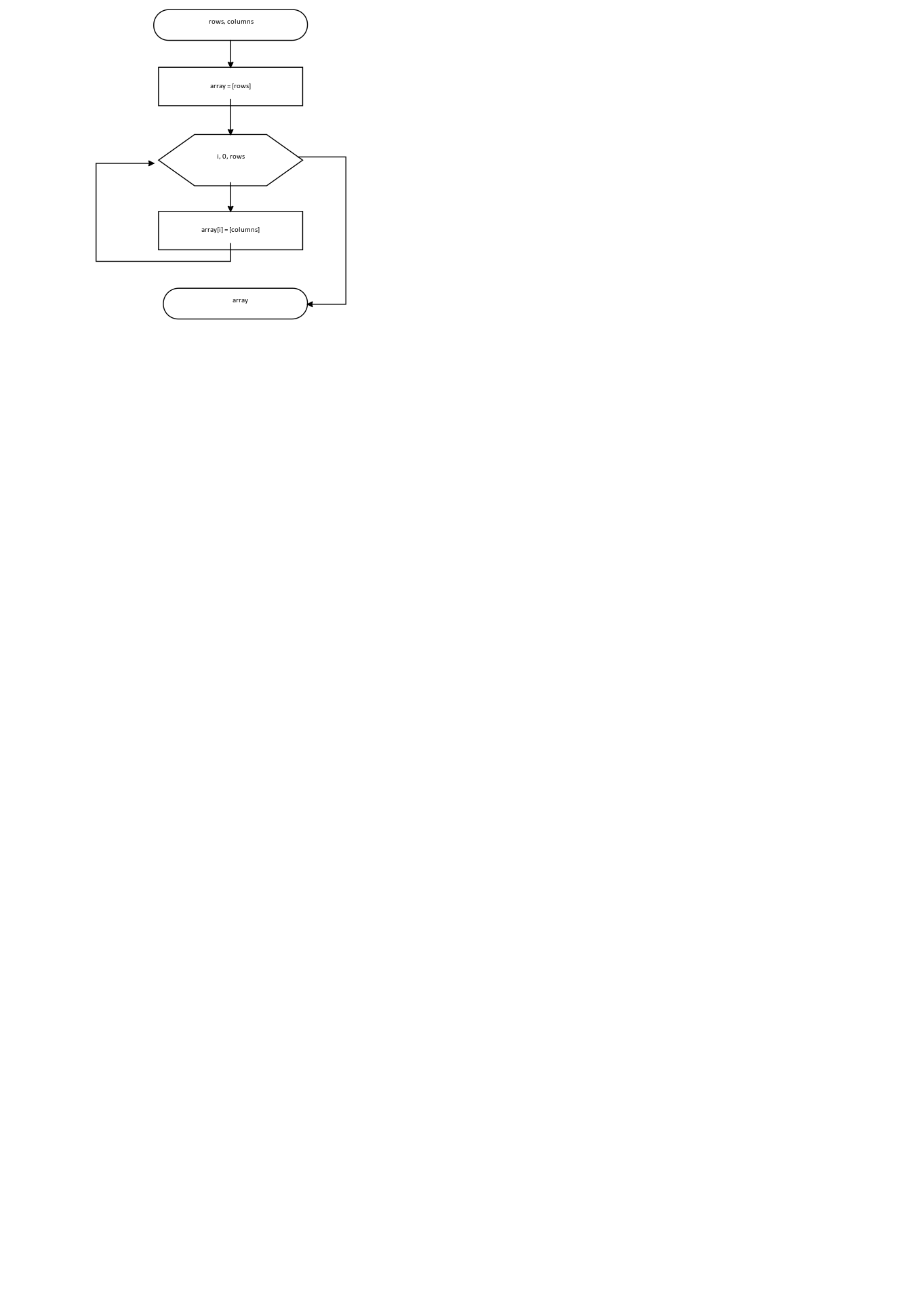
**Рисунок 2 – Блок-схема функции get\_size**



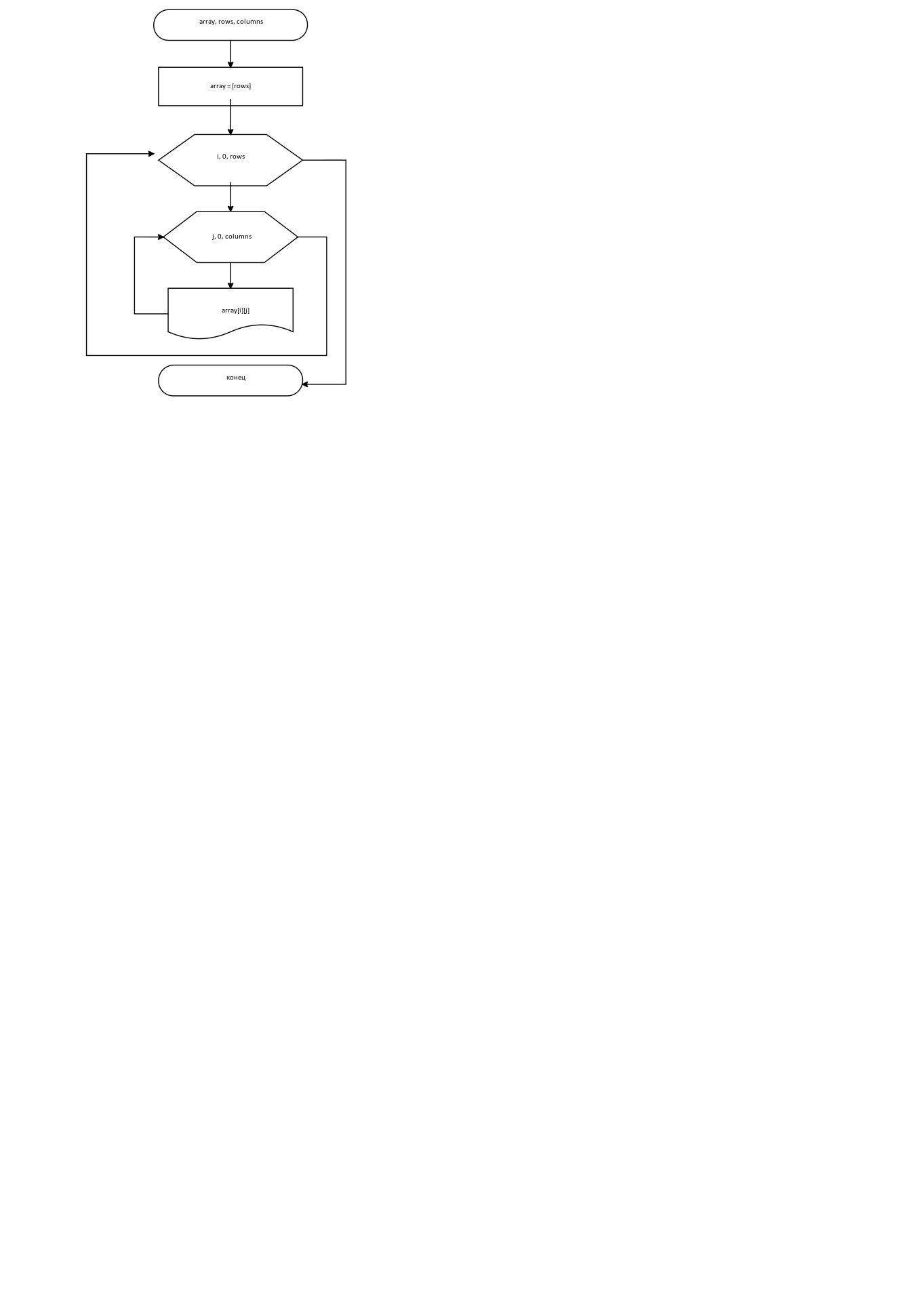
**Рисунок 3 - Блок-схема функции fill\_manual\_array**



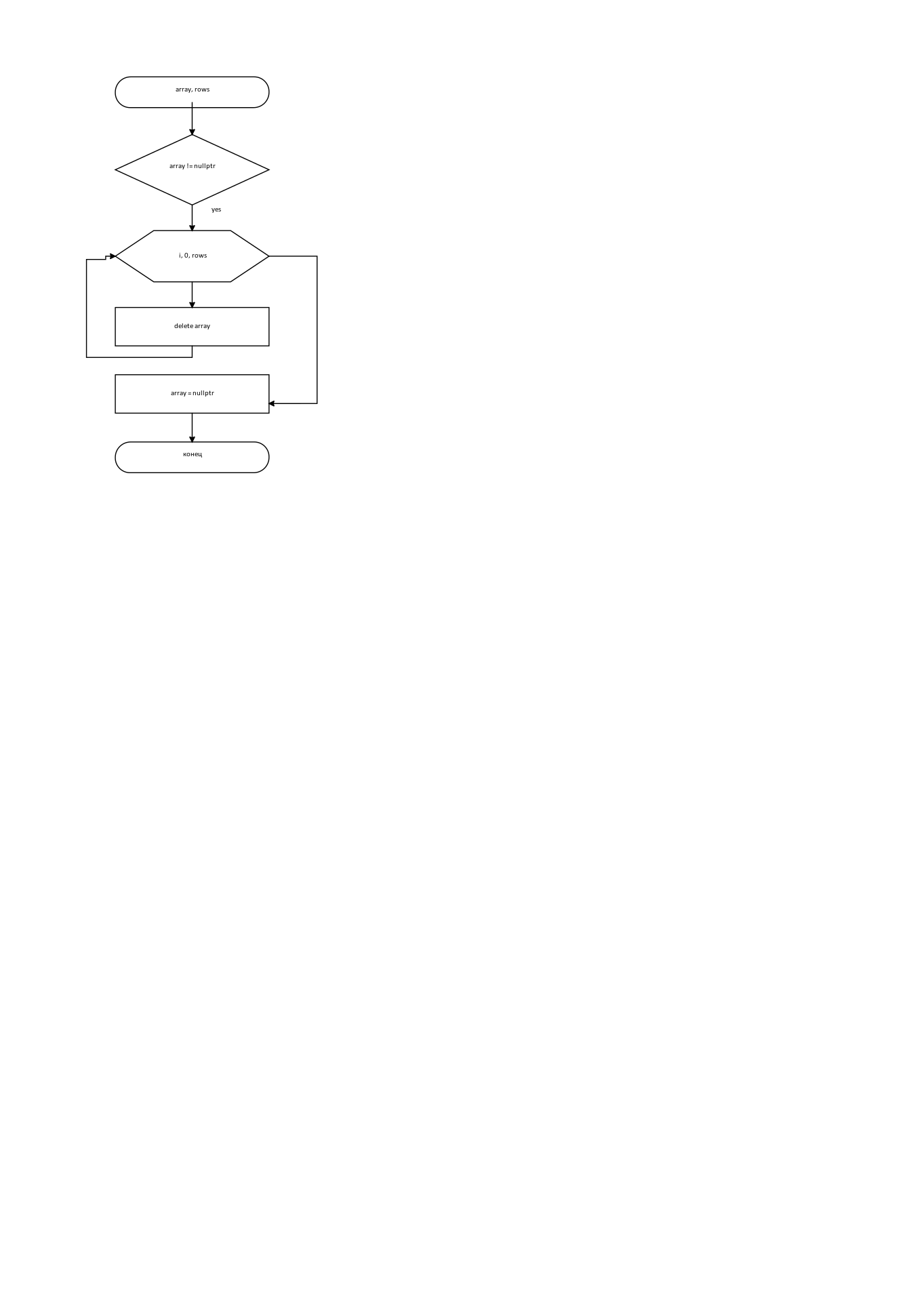
**Рисунок 4 - Блок-схема функции fill\_random\_array**



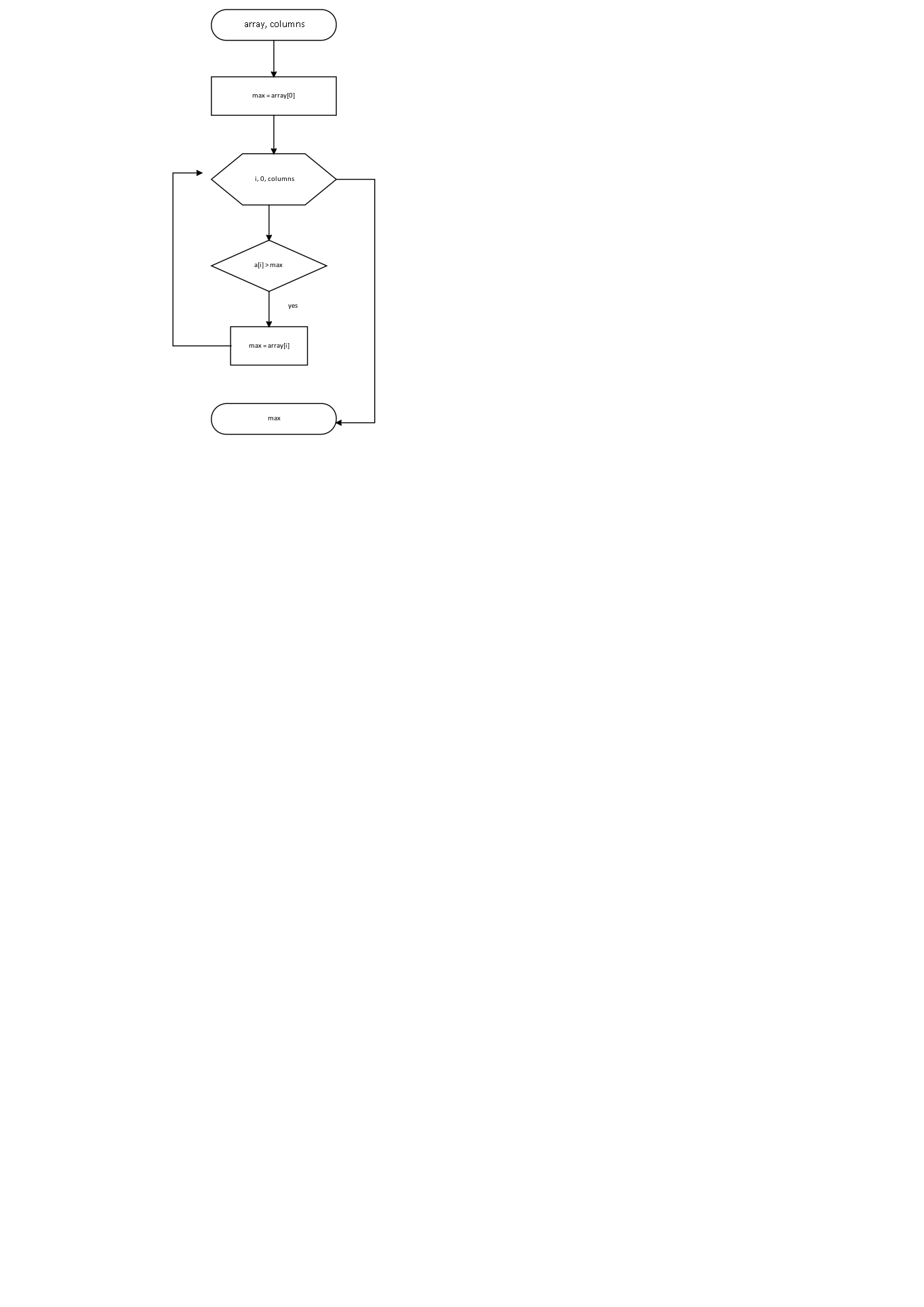
**Рисунок 5 - Блок-схема функции create\_array**



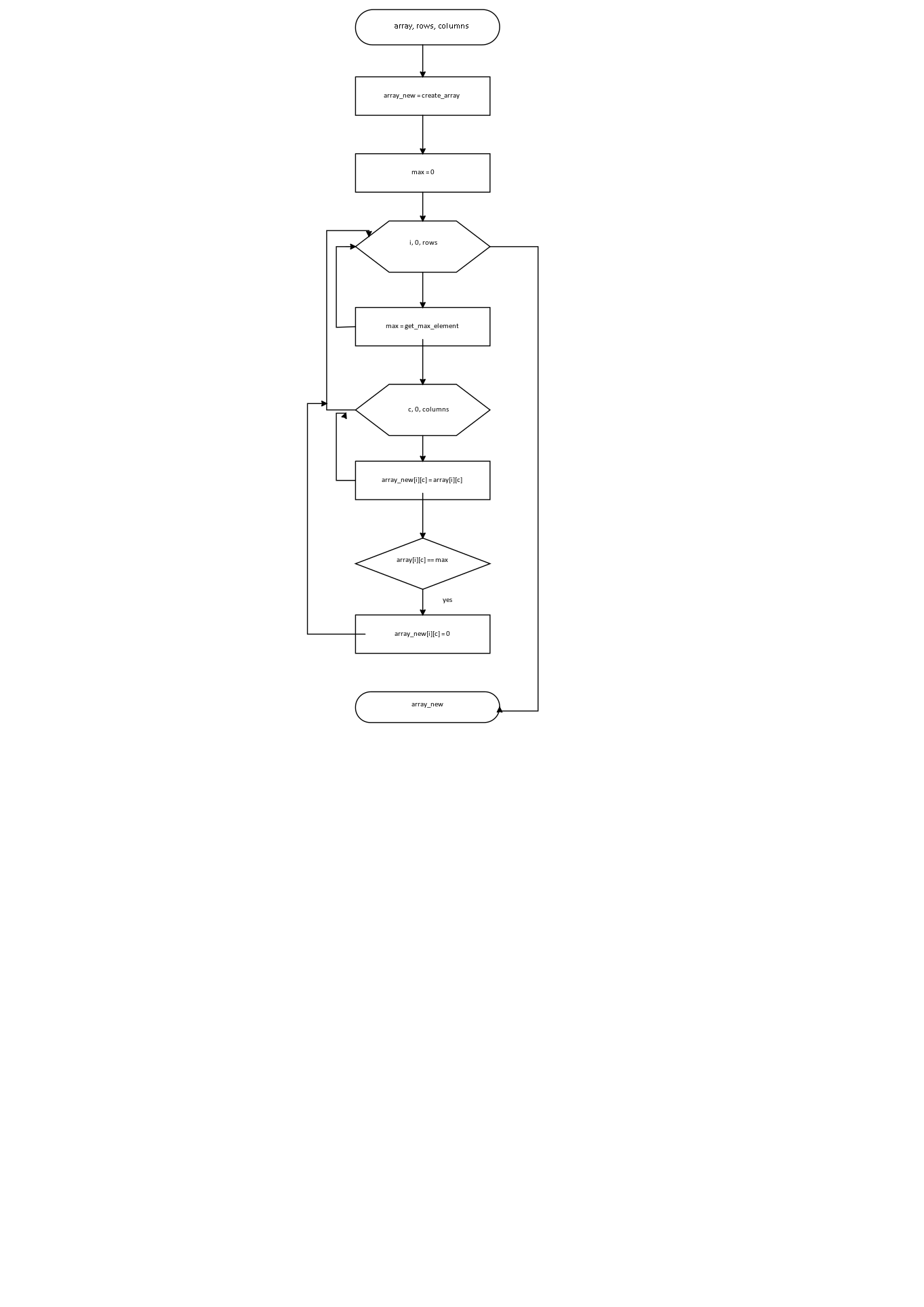
**Рисунок 6 – Блок-схема функции print\_array**



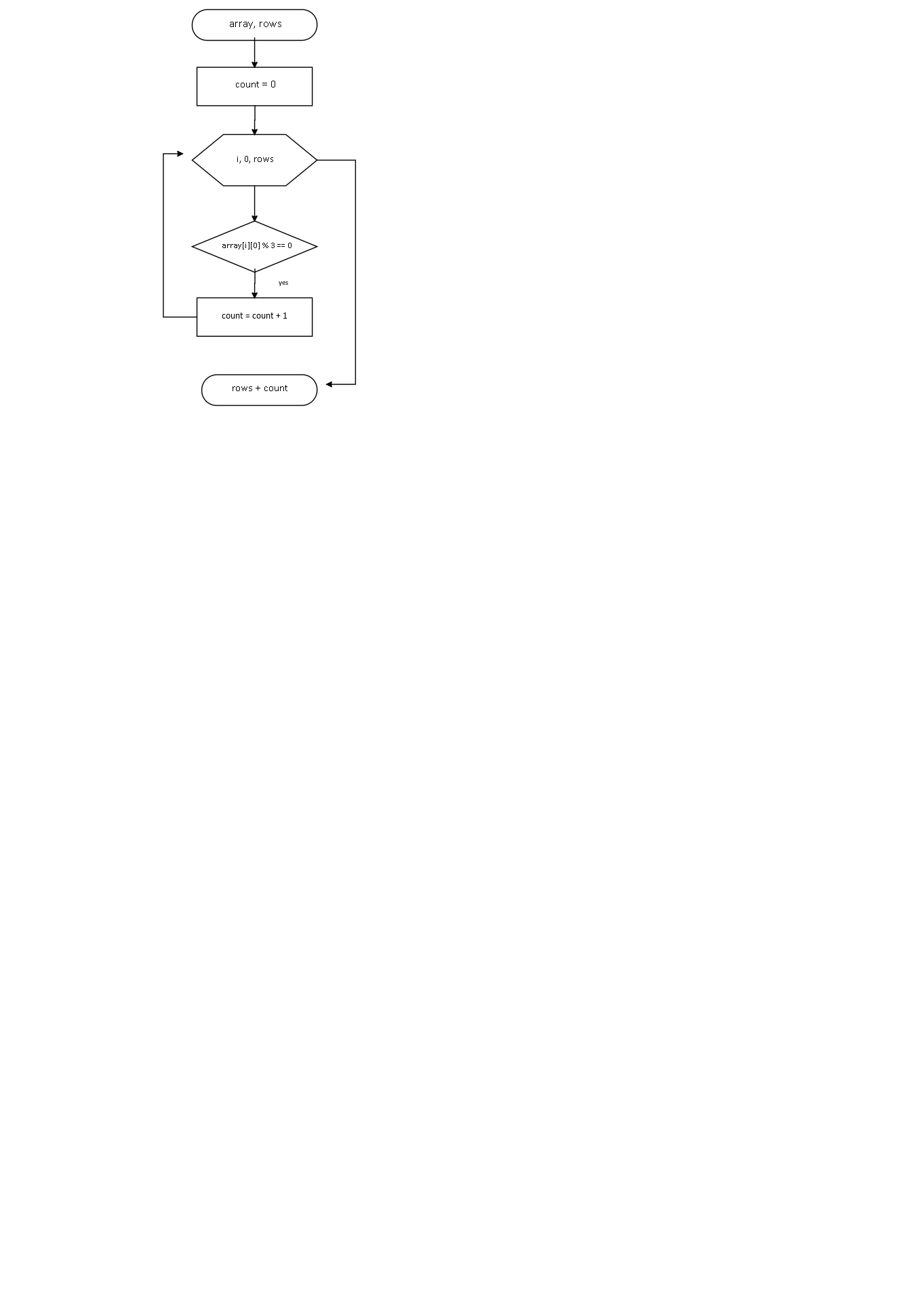
**Рисунок 7 – Блок-схема функции delete\_array**



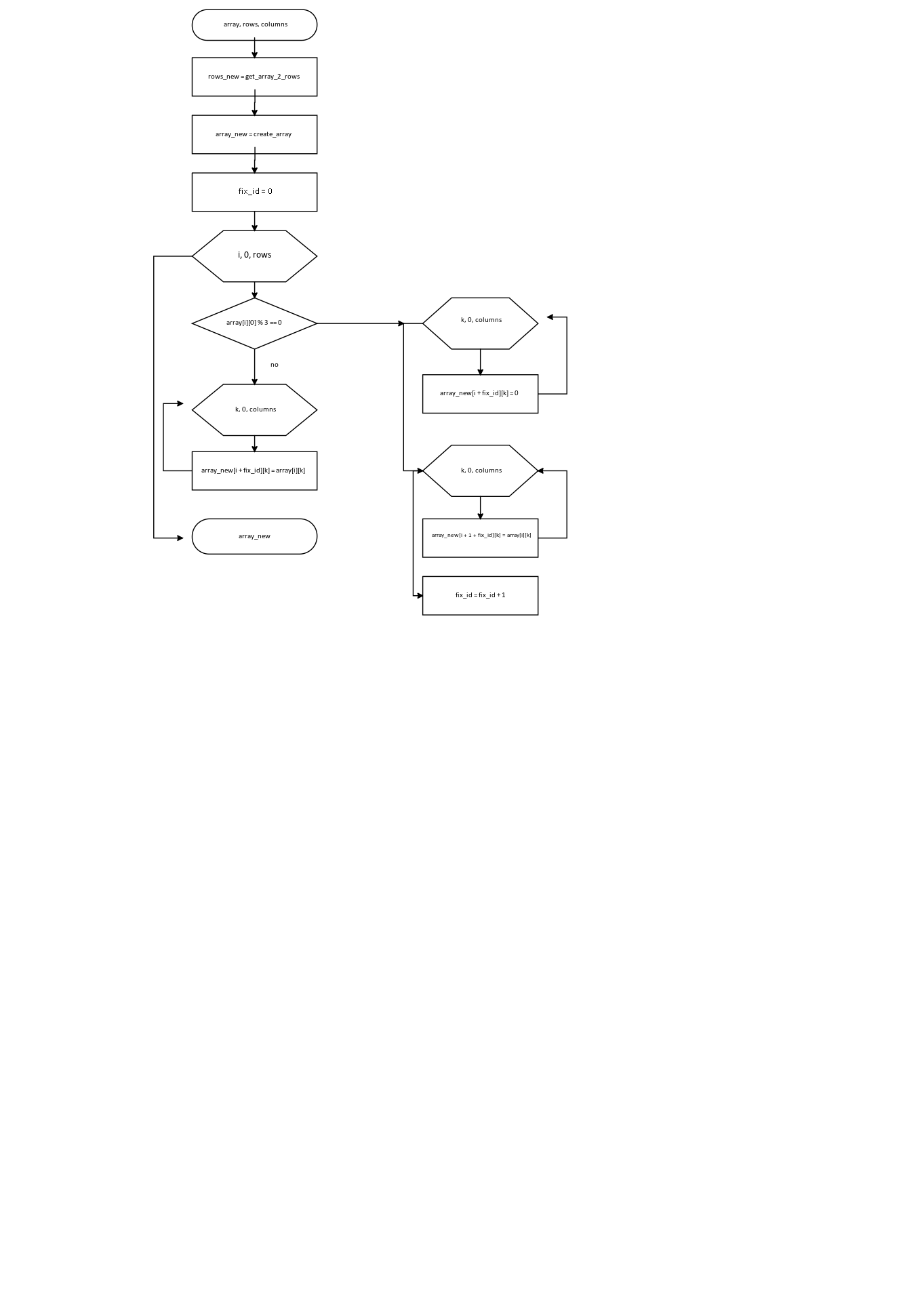
**Рисунок 8. Блок-схема функции get\_max\_element**



**Рисунок 9. Блок-схема функции task\_1**



**Рисунок 10. Блок-схема функции get\_array\_2\_rows**



**Рисунок 11. Блок-схема функции task\_2**

Решение задачи на языке программирования С++

#include <iostream>

#include <random>

using namespace std;

/\*

\*\brief Выбор способа заполнение массива

\*\param random случайное заполнение массива

\*\param manual ручное заполнение массива

\*/

enum class fill\_array

{

random,

manual

};

/\*

\*\brief Функция считывает размер массива

\*\param message выводит сообщение для пользователя

\*\return возвращает размеры массива

\*/

size\_t get\_size(const string& message);

/\*

\*\brief Функция заполняет массив случайными элементами

\*\param rows количество строк массива

\*\param columns количество столбцов массива

\*\param min\_value минимальное значение генерируемого элемента

\*\param max\_value максимальное значение генерируемого элемента

\*\return возвращает указатель на массив заполненный случайными элементами

\*/

int\*\* fill\_random\_array(const size\_t rows, const size\_t columns, const int min\_value, const int max\_value);

/\*

\*\brief Функция заполняет массив элементами введенными пользователем

\*\param rows количество строк массива

\*\param columns количество столбцов массива

\*\return возвращает указатель на массив заполненный пользователем

\*/

int\*\* fill\_manual\_array(const size\_t rows, const size\_t columns);

/\*

\*\brief Функция создающая массив

\*\param rows количество строк массива

\*\param columns количество столбцов массива

\*\return возвращает указатель на созданный массив

\*/

int\*\* create\_array(const size\_t rows, const size\_t columns);

/\*

\*\brief Функция выводящая массив на экран

\*\param array указатель на массив

\*\param rows количество строк массива

\*\param columns количество столбцов массива

\*/

void print\_array(int\*\* array, const size\_t rows, const size\_t columns);

/\*

\*\brief Функция возвращающая максимальный элемент одномерного массива

\*\param array указатель на массив

\*\param columns количество столбцов массива

\*\return возвращает максимальный элемент

\*/

int get\_max\_element(int\* array, const size\_t columns);

/\*

\*\brief Функция для подсчета первого задания

\*\param array исходный массив

\*\param rows количество строк массива

\*\param columns количество столбцов массива

\*\return возвращает указатель на массив для задания 1

\*/

int\*\* task\_1(int\*\* array, const size\_t rows, const size\_t columns);

/\*

\*\brief Функция для подсчета второго задания

\*\param array исходный массив

\*\param rows количество строк массива

\*\param columns количество столбцов массива

\*\return возвращает указатель на массив для задания 1

\*/

int\*\* task\_2(int\*\* array, const size\_t rows, const size\_t columns);

/\*

\*\brief Функция считает количест во строк в массиве для задания 2

\*\param array указатель на массив

\*\param rows количество строк массива

\*\param columns количество столбцов массива

\*\return возвращает колличество строк в массиве для задания 2

\*/

size\_t get\_array\_2\_rows(int\*\* array, const size\_t rows);

/\*

\*\brief Функция удаляющая массив

\*\param array указатель на массив

\*\param rows количество строк массива

\*/

void delete\_array(int\*\*& array, const size\_t rows);

/\*

\*\brief Точка входа в программу

\*\return возвращает 0 в случае успеха

\*/

int main()

{

setlocale(LC\_ALL, "Rus");

int rows = get\_size("Введите количество строк: \n");

int columns = get\_size("Введите количество столбцов: \n");;

const int min\_value = -100;

const int max\_value = 100;

int\*\* array = create\_array(rows, columns);

cout << "Как следует заполнить массив?" << "\n" << static\_cast<int>(fill\_array::random) << '\t' << "Случайно" << "\n" << static\_cast <int>(fill\_array::manual) << '\t' << "Пользовательский ввод" << endl;

int input = 0;

cin >> input;

const auto choice = static\_cast<fill\_array>(input);

switch (choice)

{

case(fill\_array::random):

{

array = fill\_random\_array(rows, columns, min\_value, max\_value);

break;

}

case(fill\_array::manual):

{

array = fill\_manual\_array(rows, columns);

break;

}

default:

{

cout << "error" << endl;

}

}

int\*\* array\_task\_1 = task\_1(array, rows, columns);

print\_array(array, rows, columns);

cout << "\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_Заменим максимальный элемент строки на 0\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_" << endl;

print\_array(array\_task\_1, rows, columns);

int\*\* array\_task\_2 = task\_2(array, rows, columns);

cout << "\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_Вставим строку из нулей - перед строкой, первый элемент котрой делится на 3\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_" << endl;

size\_t array\_2\_rows = get\_array\_2\_rows(array, rows);

print\_array(array\_task\_2, array\_2\_rows, columns);

delete\_array(array, rows);

delete\_array(array\_task\_1, rows);

delete\_array(array\_task\_2, array\_2\_rows);

return 0;

}

size\_t get\_size(const string& message)

{

int size = -1;

cout << message;

cin >> size;

if (size < 0)

{

throw out\_of\_range("Incorrect size. Value has to be greater or equal zero.");

}

return size;

}

int\*\* fill\_manual\_array( const size\_t rows,const size\_t columns)

{

int\*\* array = create\_array(rows, columns);

for (size\_t i = 0; i < rows; i++)

{

for (size\_t j = 0; j < columns; j++)

{

int l = 0;

cin >> l;

array[i][j] = l;

}

}

return array;

}

int\*\* fill\_random\_array(const size\_t rows, const size\_t columns, const int min\_value, const int max\_value)

{

int\*\* array = create\_array(rows, columns);

random\_device rd;

mt19937 gen(rd());

uniform\_int\_distribution<> uniformIntDistribution(min\_value, max\_value);

for (size\_t i = 0; i < rows; i++)

{

for (size\_t j = 0; j < columns; j++)

{

array[i][j] = uniformIntDistribution(gen);

}

}

return array;

}

int\*\* create\_array( const size\_t rows, const size\_t columns)

{

int\*\* array = new int\* [rows];

for (size\_t i = 0; i < rows; i++)

{

array[i] = new int[columns];

}

return array;

}

void print\_array(int\*\* array, const size\_t rows, const size\_t columns)

{

for (size\_t i = 0; i < rows; i++)

{

for (size\_t j = 0; j < columns; j++)

{

cout << array[i][j] << "\t";

}

cout << "\n";

}

}

int get\_max\_element(int\* array, const size\_t columns)

{

int max = array[0];

for (size\_t i = 0; i < columns; i++)

{

if (array[i] > max)

{

max = array[i];

}

}

return max;

}

int\*\* task\_1(int\*\* array, const size\_t rows, const size\_t columns)

{

int\*\* array\_new = create\_array(rows, columns);

int max = 0;

for (size\_t i = 0; i < rows; i++)

{

max = get\_max\_element(array[i], columns);

for (size\_t c = 0; c < columns; c++)

{

array\_new[i][c] = array[i][c];

if (array[i][c] == max)

{

array\_new[i][c] = 0;

}

}

}

return array\_new;

}

size\_t get\_array\_2\_rows(int\*\* array, const size\_t rows)

{

size\_t count = 0;

for (size\_t i = 0; i < rows; i++)

{

if (array[i][0] % 3 == 0)

{

count++;

}

}

return rows + count;

}

int\*\* task\_2(int\*\* array, const size\_t rows, const size\_t columns)

{

size\_t rows\_new = get\_array\_2\_rows( array, rows);

int\*\* array\_new = create\_array(rows\_new, columns);

size\_t fix\_id = 0;

for (size\_t i = 0; i < rows; i++)

{

if (array[i][0] % 3 == 0)

{

for (size\_t k = 0; k < columns; k++)

{

array\_new[i + fix\_id][k] = 0;

}

for (size\_t k = 0; k < columns; k++)

{

array\_new[i + 1 + fix\_id][k] = array[i][k];

}

fix\_id++;

}

else

{

for (size\_t k = 0; k < columns; k++)

{

array\_new[i + fix\_id][k] = array[i][k];

}

}

}

return array\_new;

}

void delete\_array(int\*\*& array, const size\_t rows)

{

if (array != nullptr)

{

for (size\_t i = 0; i < rows; i++)

{

delete[] array[i];

}

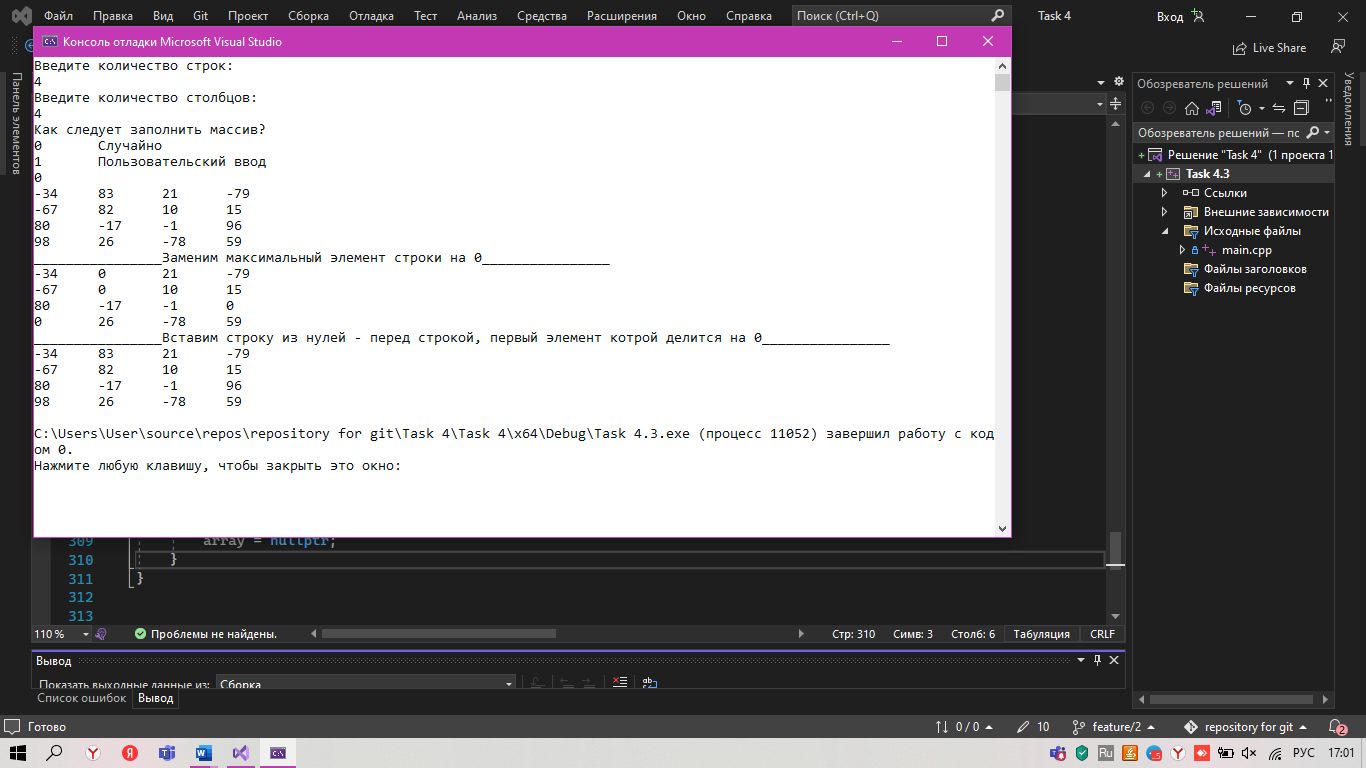
delete[] array;

array = nullptr;

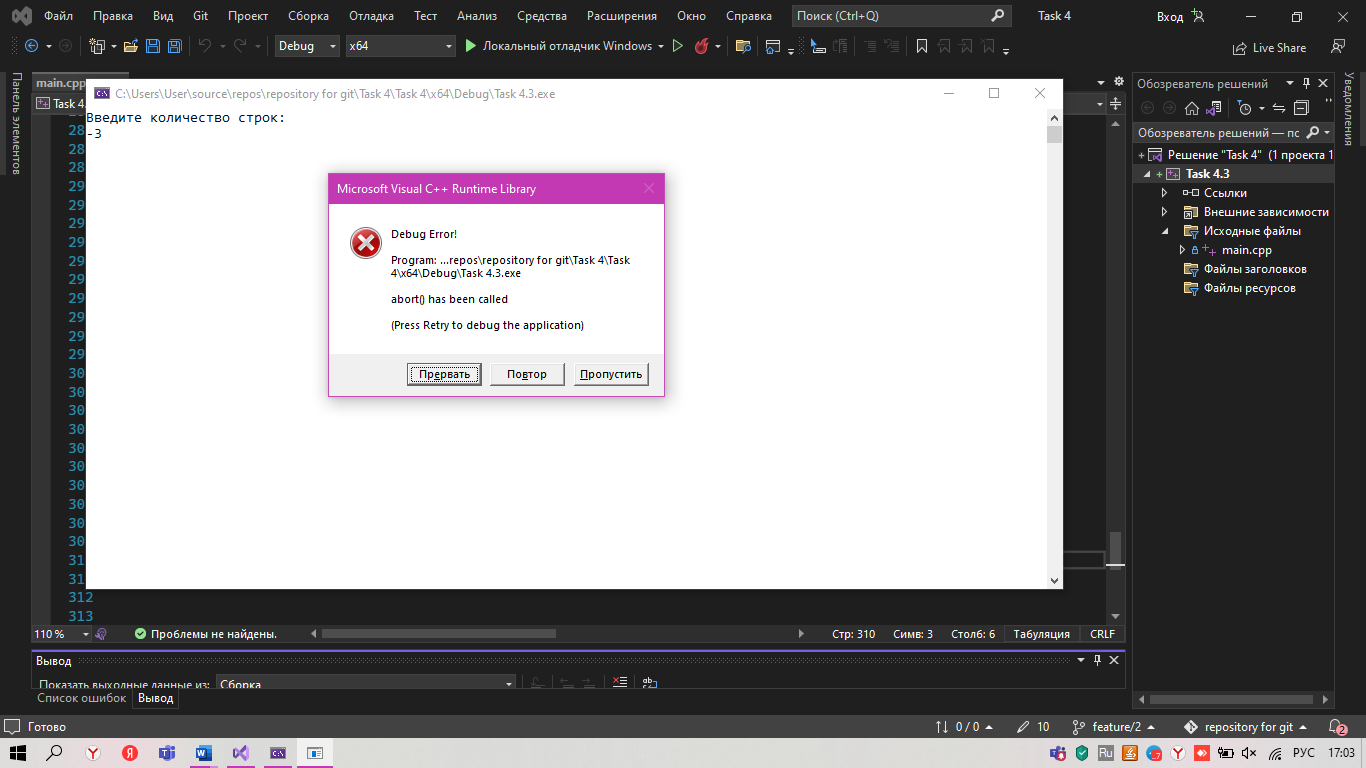
}

}

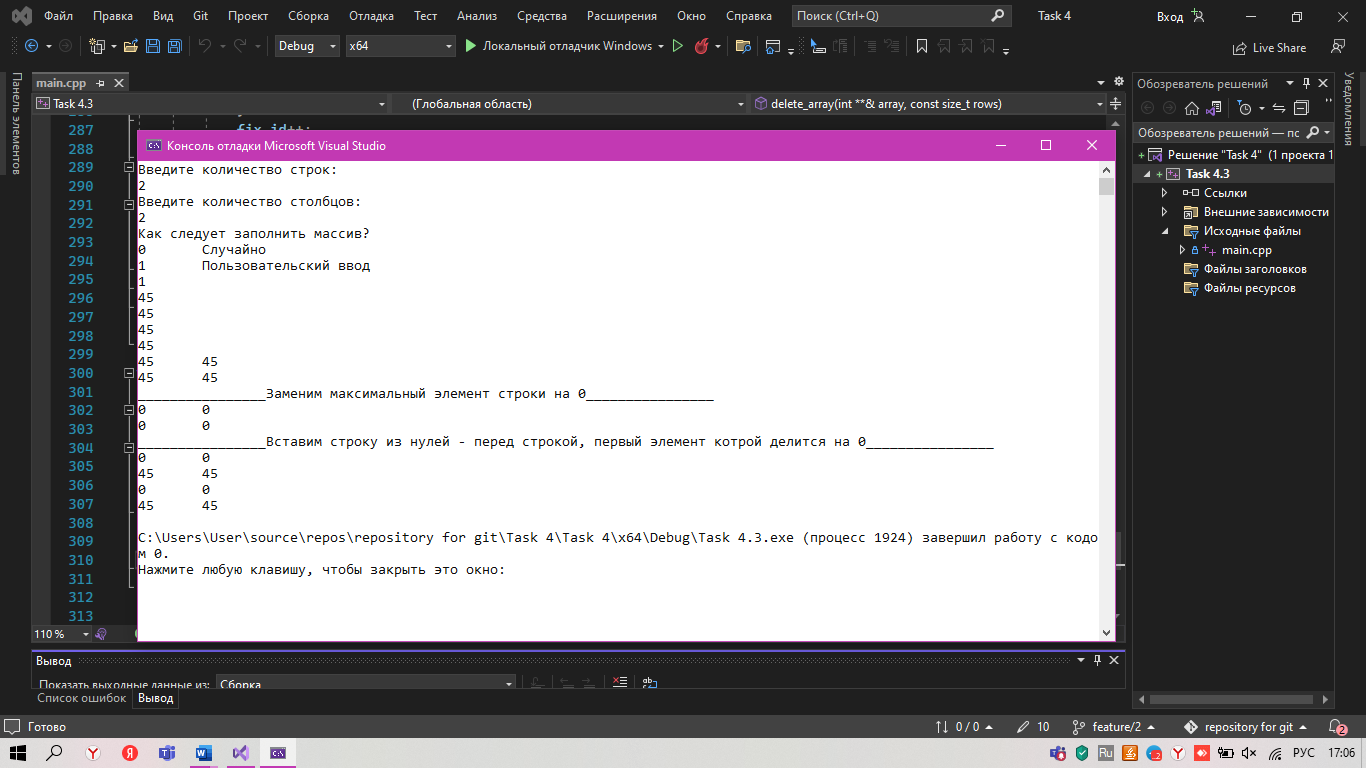
**3. Решение тестовых примеров**



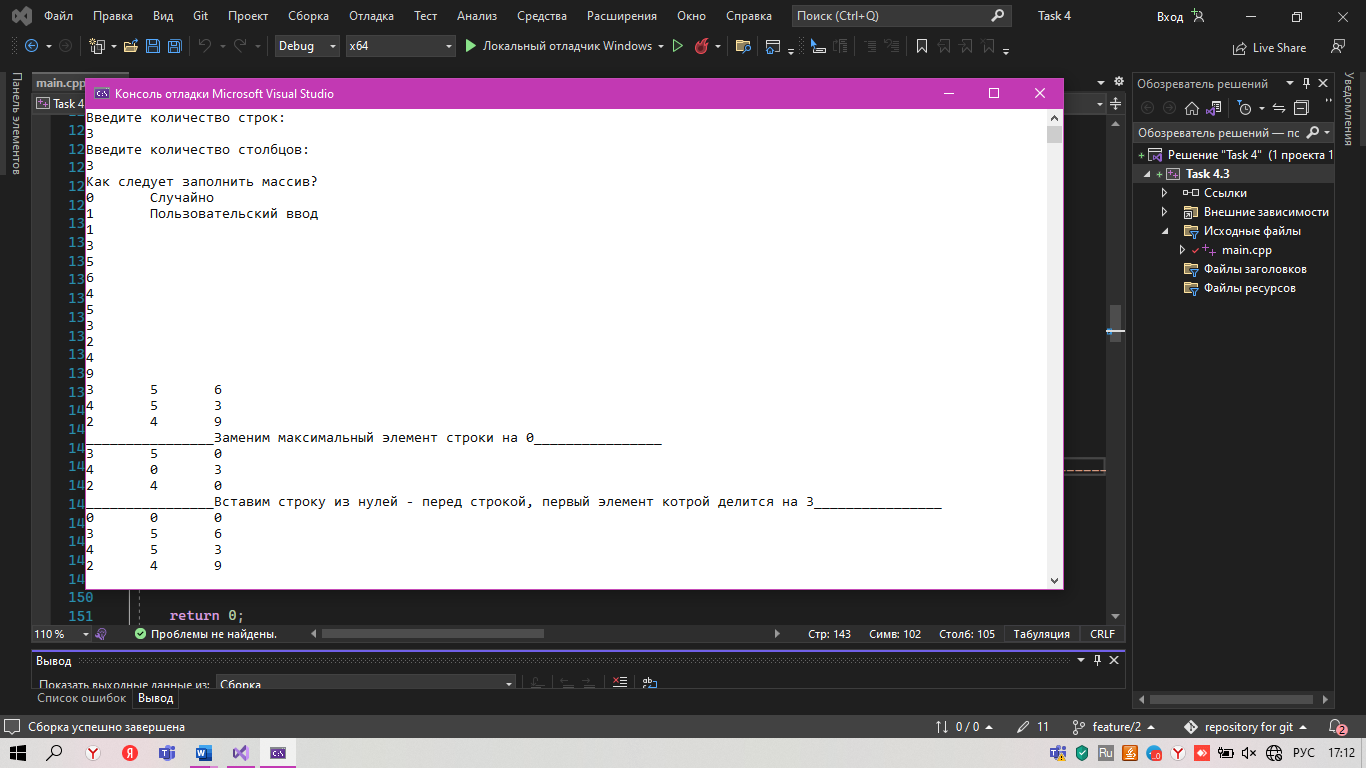
**Рисунок 12. Решение тестового примера**



**Рисунок 13. Отрицательный размер массива**



**Рисунок 14. Заполнение массива максимальным и кратным 3 элементом**



**Рисунок 15. Ручное заполнение массива**

**4. Зачет задания в GitHub**

