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 ЛАБОРАТОРНОЙ РАБОТЕ №4

По дисциплине «Языки программирования»

ВАРИАНТ 9

Выполнил: ст. гр. ТКИ - 141

Ткачев Вадим Евгеньевич

Проверил: к.т.н., доц. Васильева М. А.

(Проверил: к.т.н, доц. Балакина Е. П.)

Москва 2024

Оглавление

[1. Задание 4.1 3](#__RefHeading___Toc2718_3174983259)

[1.1. Формулировка задания 3](#__RefHeading___Toc2720_3174983259)

[1.2. Блок-схема алгоритма 4](#__RefHeading___Toc2722_3174983259)

[1.3. Код задания 4.1 13](#__RefHeading___Toc2724_3174983259)

[1.4. Решение тестового примера 19](#__RefHeading___Toc2726_3174983259)

[1.5. Зачёт задания в GitHub 20](#__RefHeading___Toc2728_3174983259)

[2. Задание 4.2 21](#__RefHeading___Toc2730_3174983259)

[2.1. Формулировка задания 21](#__RefHeading___Toc2732_3174983259)

[2.2. Блок-схема алгоритма 22](#__RefHeading___Toc2734_3174983259)

[2.3. Код задания 4.2 36](#__RefHeading___Toc2736_3174983259)

[2.4. Решение тестового примера 43](#__RefHeading___Toc2738_3174983259)

[2.5. Зачёт задания в GitHub 43](#__RefHeading___Toc2740_3174983259)

[3. Задание 4.3 44](#__RefHeading___Toc20964_2976669137)

[3.1. Формулировка задания 44](#__RefHeading___Toc20966_2976669137)

[3.2 Блок-схема алгоритма 44](#__RefHeading___Toc5987_2494015749)

[3.3 Текст программы на языке С 59](#__RefHeading___Toc8599_2494015749)

[3.4 Результаты выполнения программы 72](#__RefHeading___Toc10922_2494015749)

[3.5 Отметка о выполнении задания в системе контроля версий 73](#__RefHeading___Toc13245_2494015749)

1. Задание 4.1
   1. Формулировка задания

Создать одномерный массив 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). Составить блок-схему.

Таблица 1 –Формулировка задания 4.1

|  |  |  |
| --- | --- | --- |
| **Вариант** | **Задачи** | **Интервал** |
| **9** | 1. Найти сумму элементов, значения которых по модулю меньше 10. 2. Вывести индексы тех элементов, значения которых больше значения последующего элемента. 3. Умножить все элементы массива, кратные 3, на третий элемент массива. | [-40;40] |

* 1. Блок-схема алгоритма

Блок-схема алгоритма представлена на рисунках (Рисунок 1 - 15)

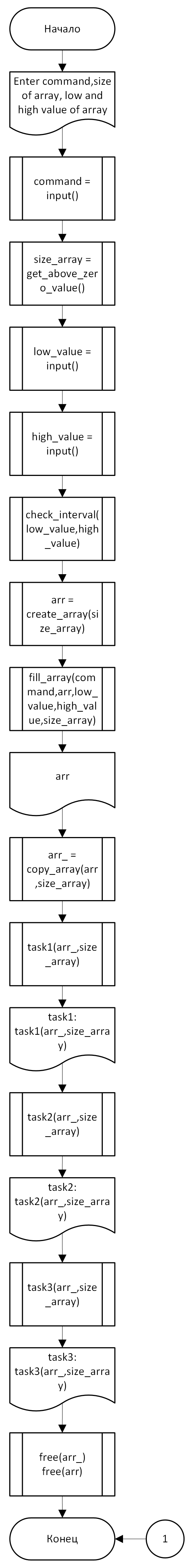


Рисунок *1* – Блок-схема функции main()

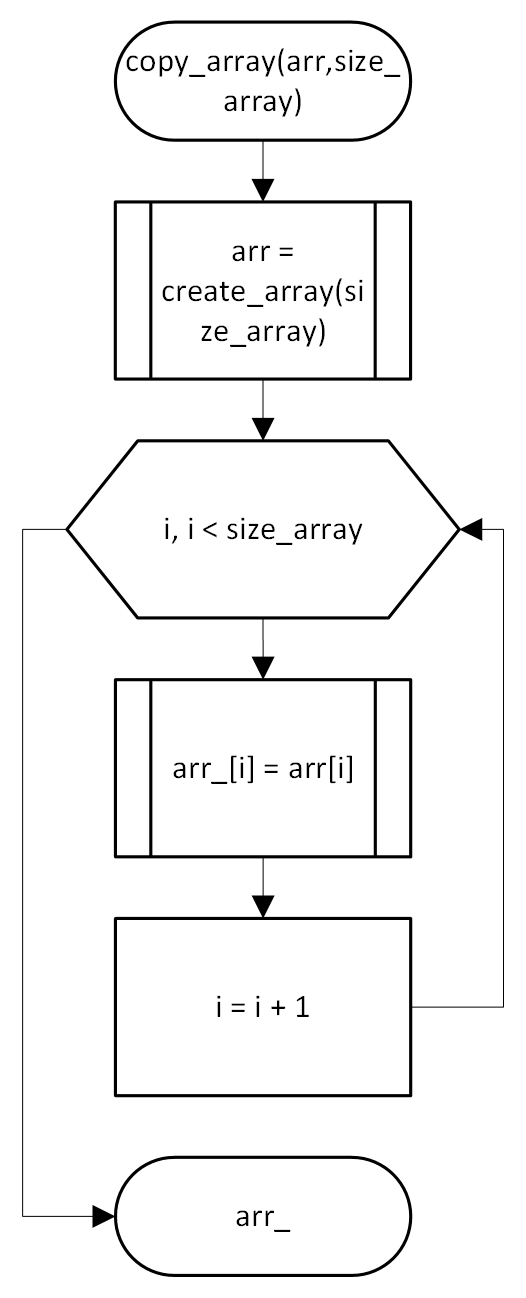


Рисунок 2 – Блок-схема функции copy\_array

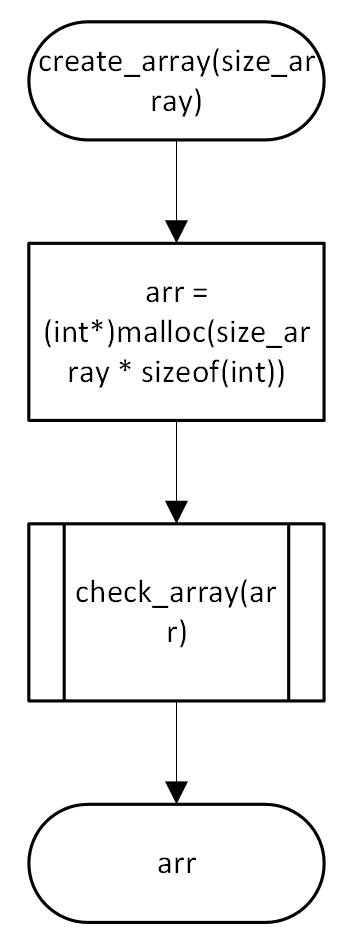


Рисунок 3 – Блок-схема функции create\_array

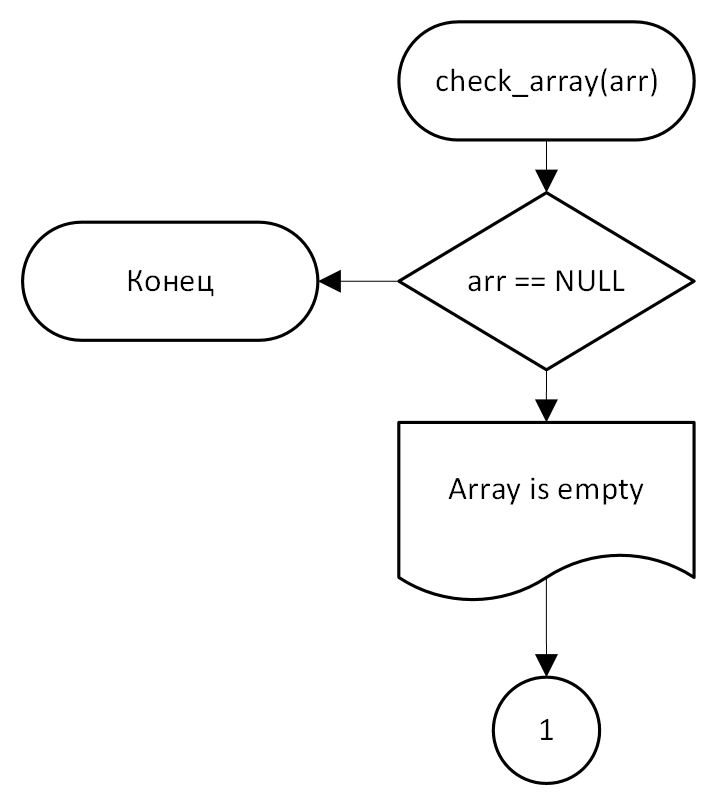


Рисунок 4 – Блок-схема функции check\_array

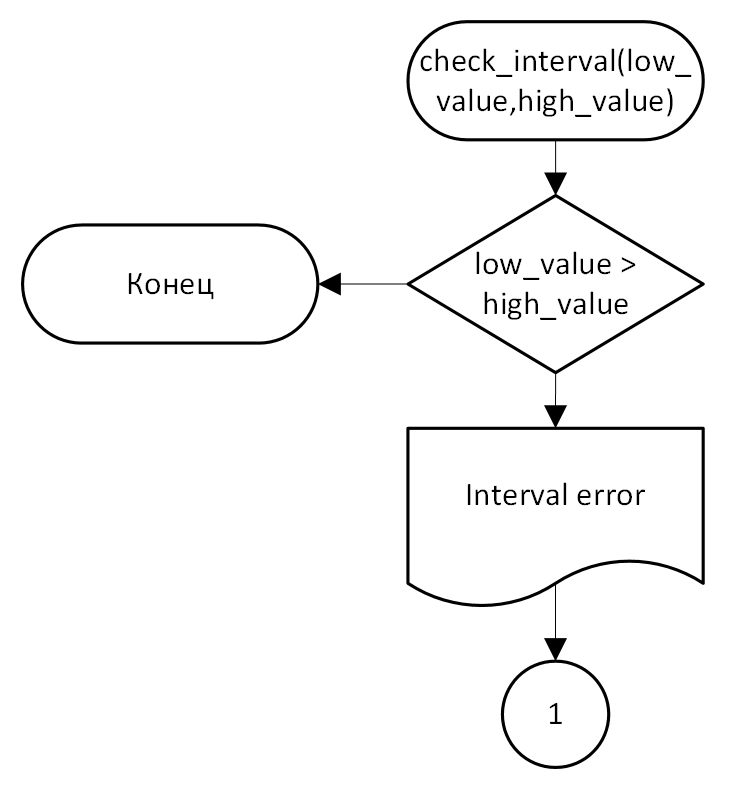


Рисунок 5 – Блок-схема функции check\_interval

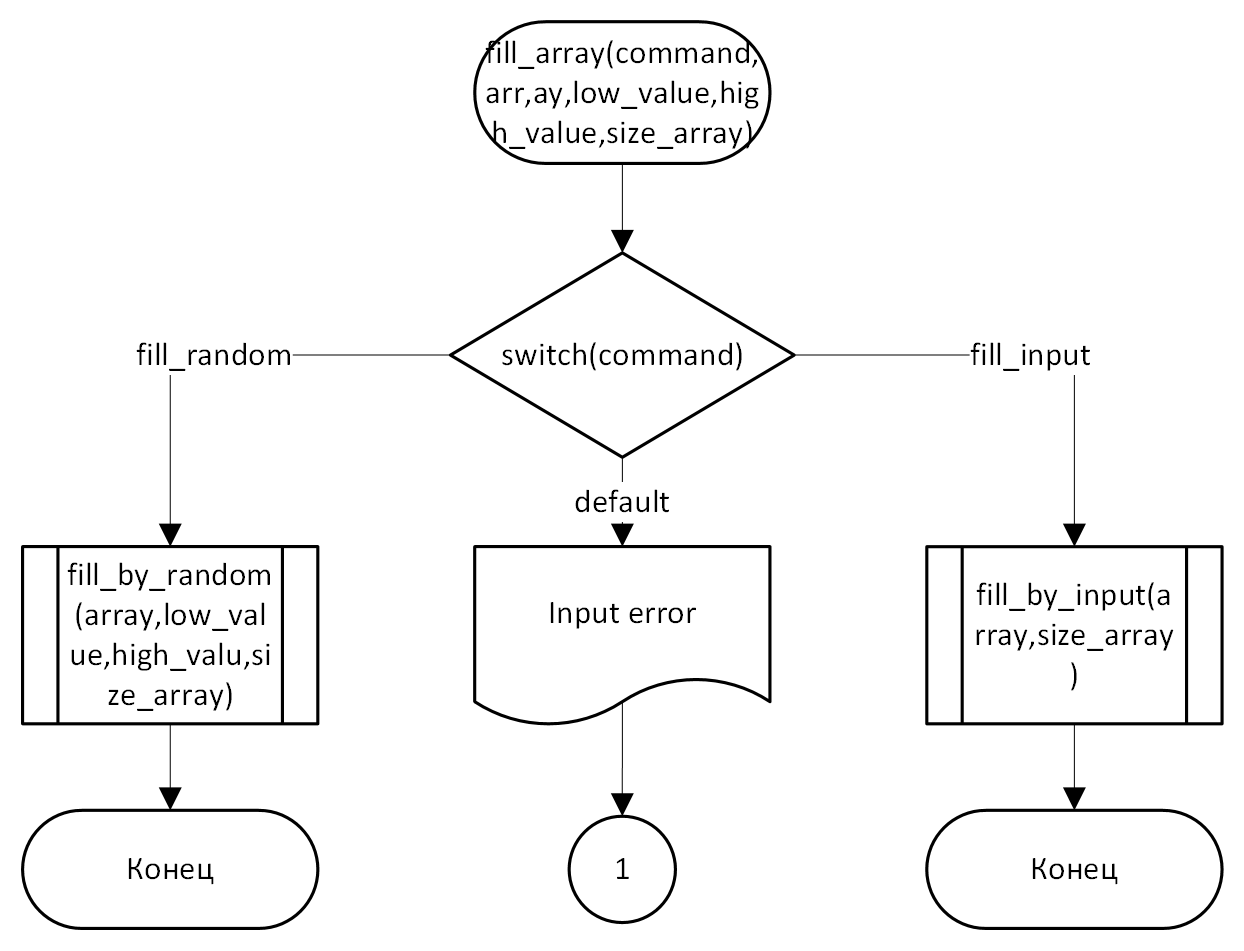


Рисунок 6 – Блок-схема функции fill\_array

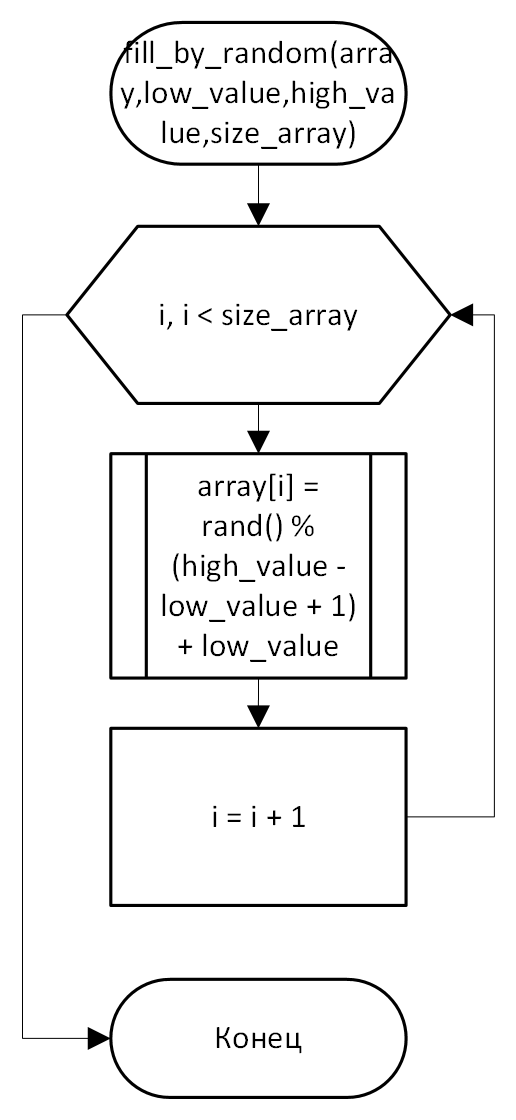


Рисунок 7 – Блок-схема функции fill\_by\_random

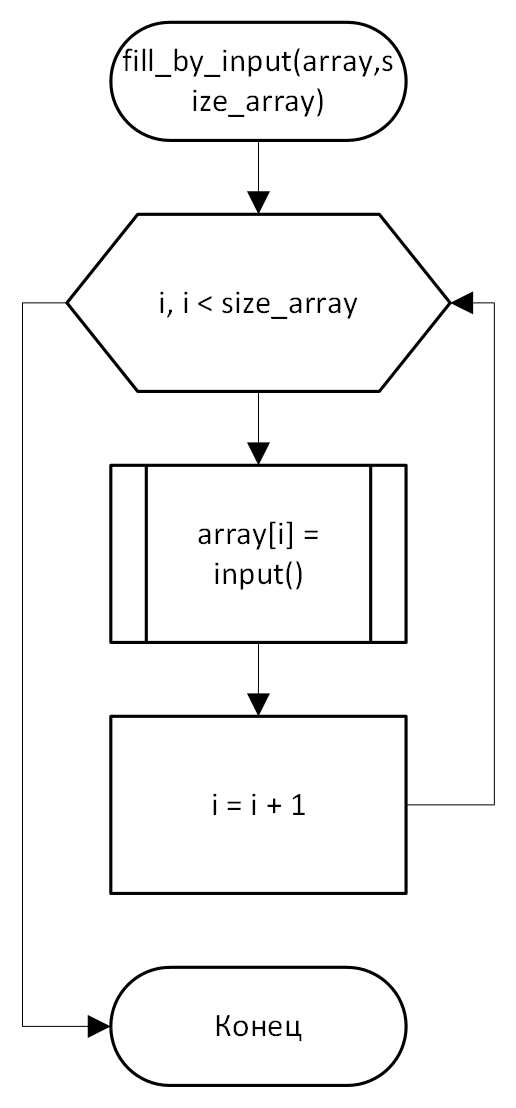


Рисунок *8* – Блок-схема функции fill\_by\_input

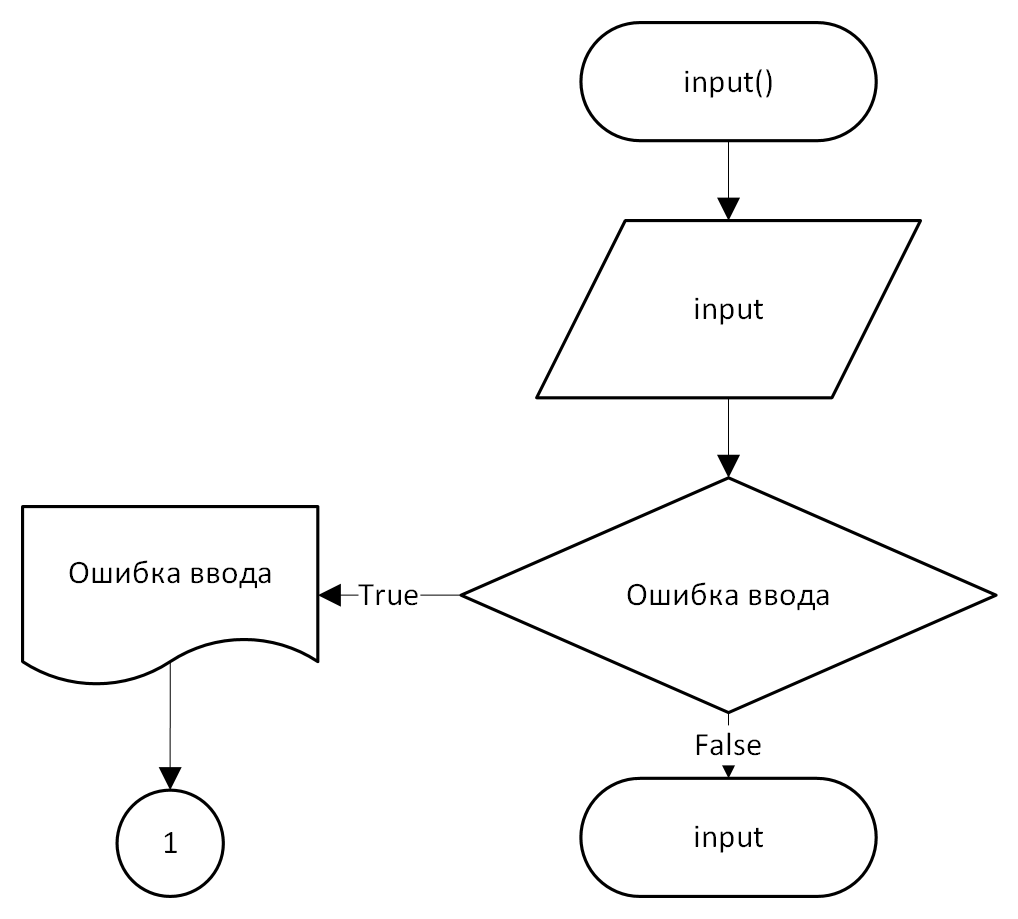


Рисунок 9 – Блок-схема функции input

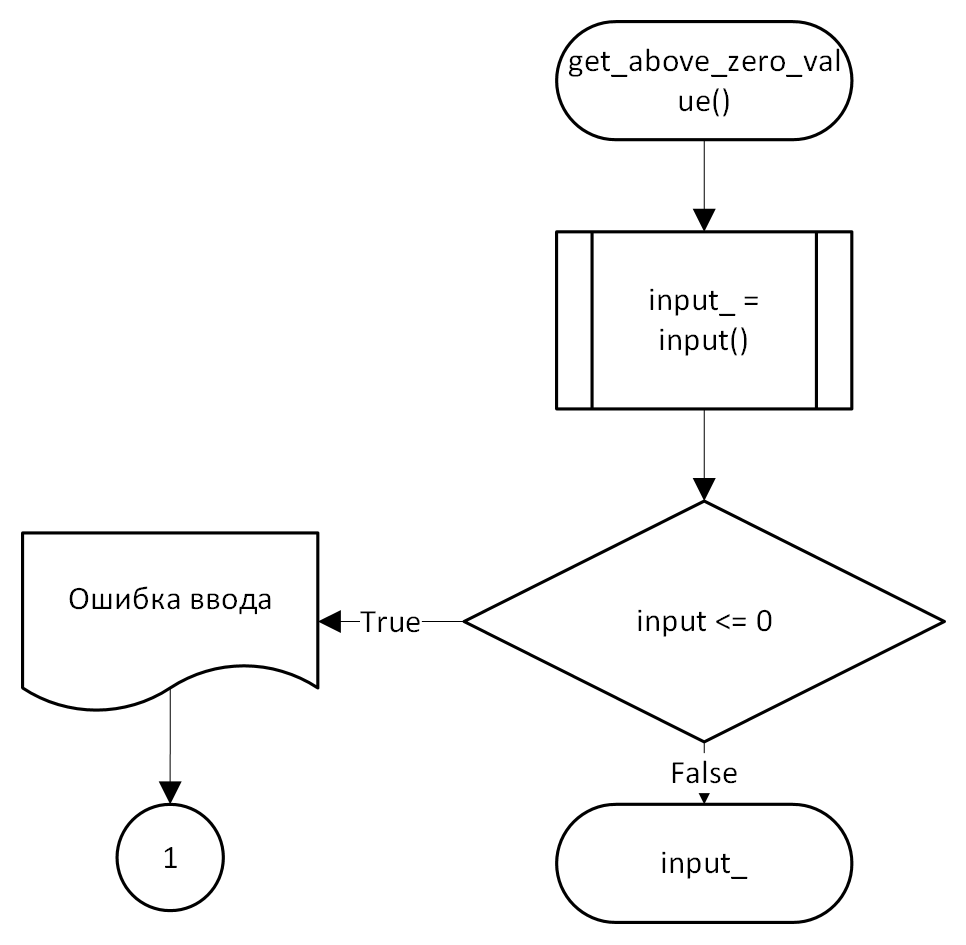


Рисунок 10 – Блок-схема функции get\_above\_zero\_value

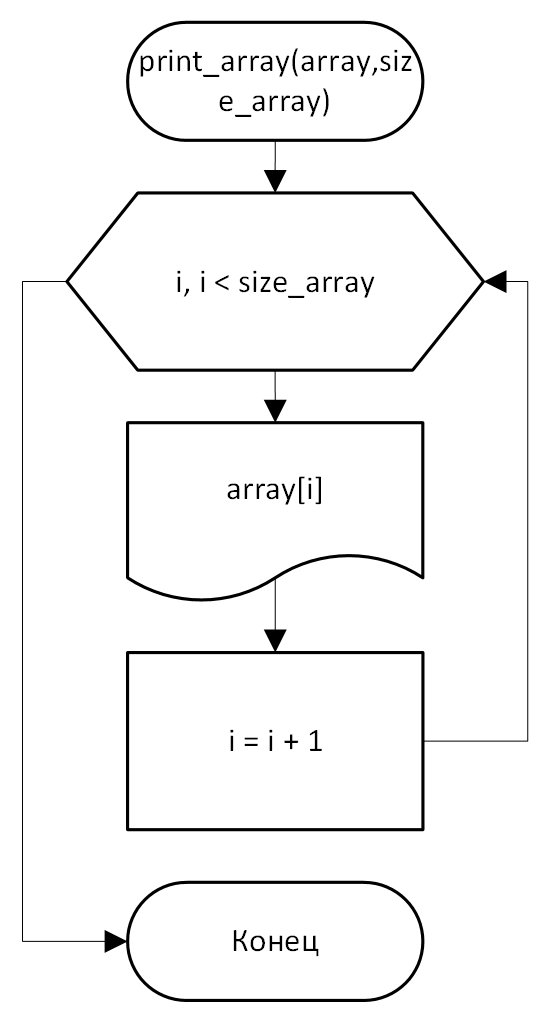


Рисунок 11 – Блок-схема функции print\_array

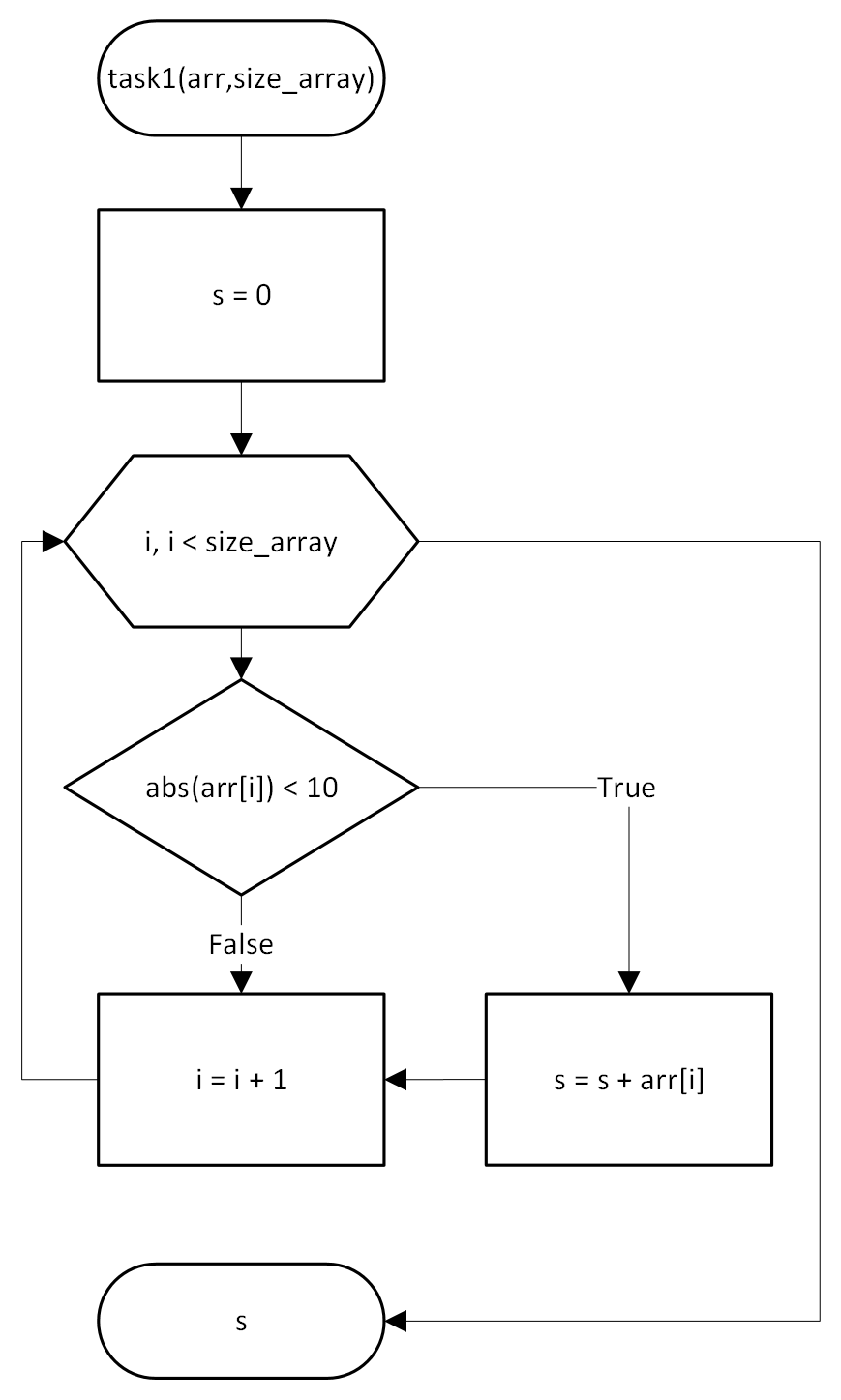
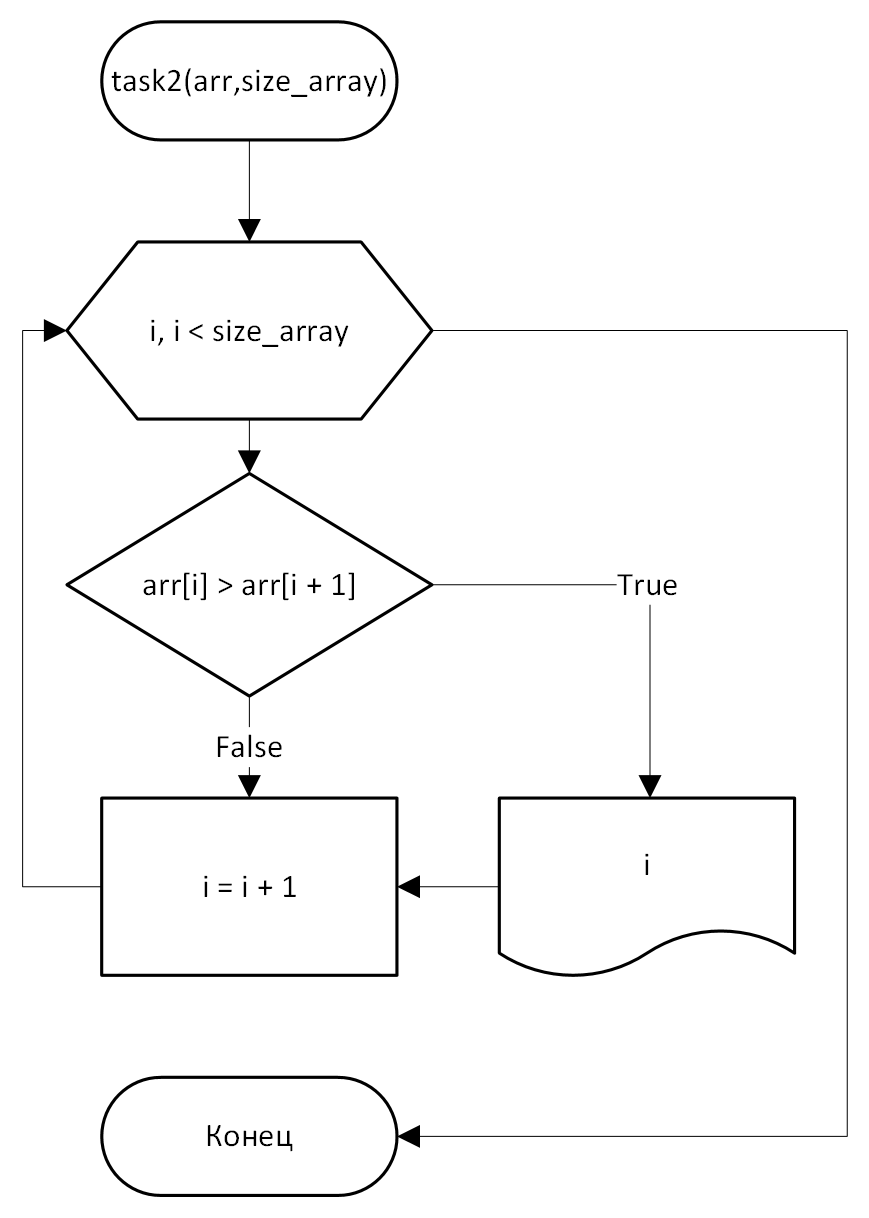
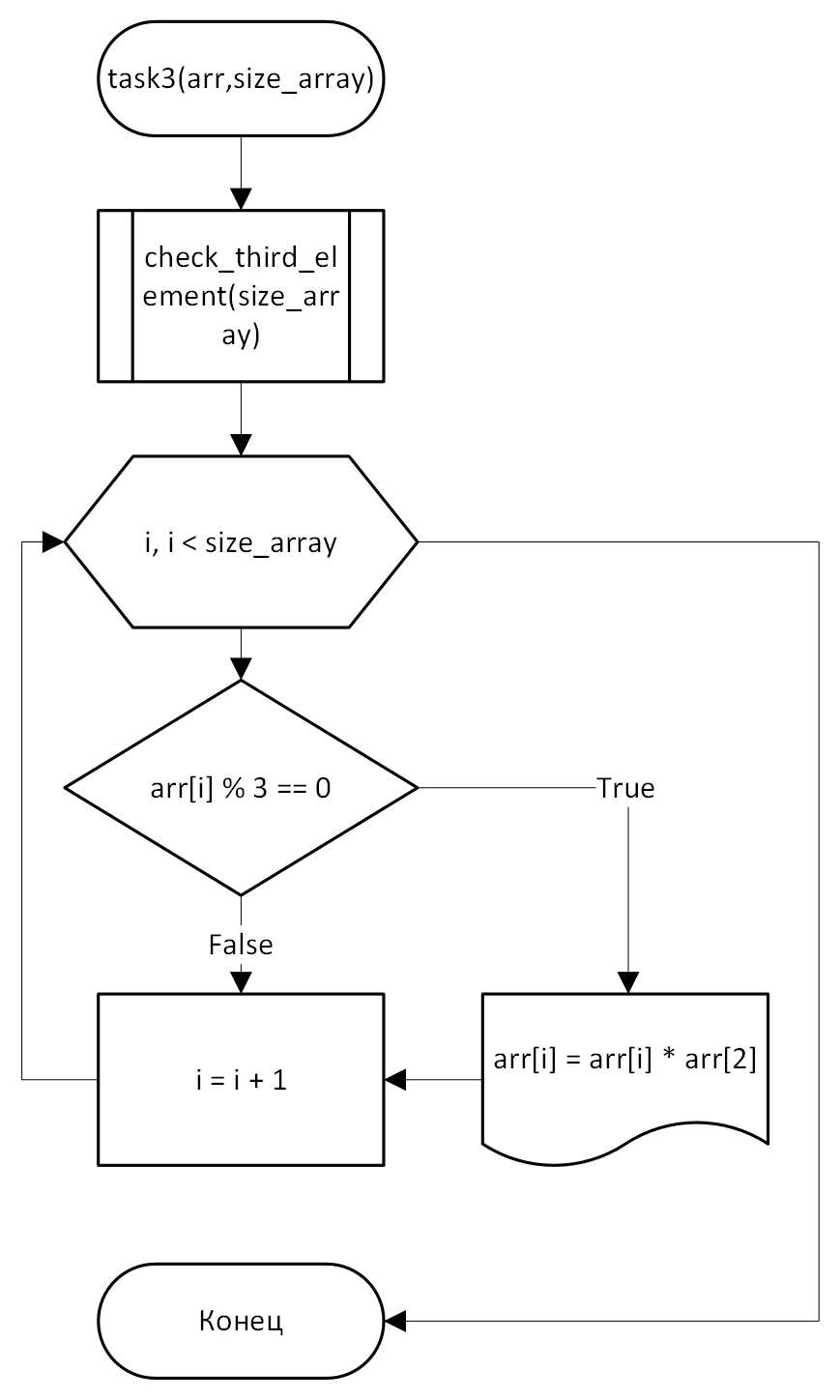
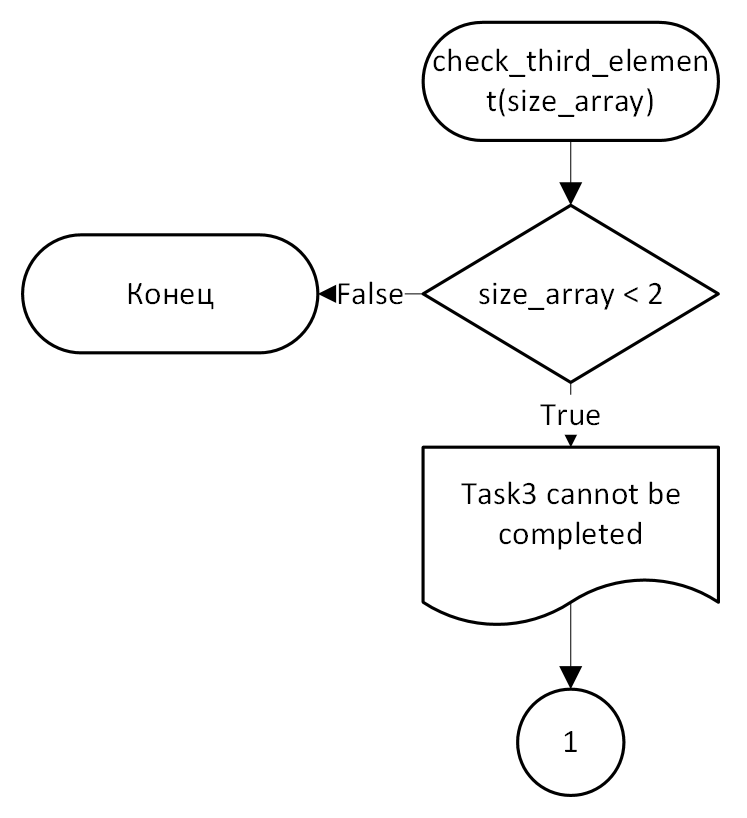


Рисунок 12 – Блок-схема функции task1

Рисунок 13 — Блок-схема функции task2

Рисунок 14 - Блок-схема функции task3

Рисунок 15 - Блок-схема функции check\_third\_element

* 1. Код задания 4.1

#include <stdio.h>

#include <stdlib.h>

#include <math.h>

#include <time.h>

/\*\*

\* @brief Функция выбора метода заполнения массива пользователем

\* @param command - переменная для выбора метода заполнения массива

\* @param array - массив, задаваемый пользователем

\* @param low\_value - начало массива

\* @param size\_array - размер массива

\* @remarks при неправильном вводе команды выводит ошибку

\*/

void fill\_array(const int command, int\* array, const int low\_value, const int high\_value, const size\_t size\_array);

/\*\*

\* @brief Функция заполнения массива интервалом, который задал пользователь

\* @param array - массив, задаваемый пользователем

\* @param size\_array - размер массива

\*/

void fill\_by\_input(int\* array, const size\_t size\_array);

/\*\*

\* @brief Функция заполнения массива случайными числами

\* @param array - массив, задаваемый пользователем

\* @param low\_value - начало массива

\* @param size\_array - размер массива

\*/

void fill\_by\_random(int\* array, const int low\_value, const int high\_value, const size\_t size\_array);

/\*\*

\* @brief Функция вывода массива

\* @param array - массив, задаваемый пользователем

\* @param size\_array - размер массива

\*/

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

/\*\*

\* @brief Функция ввода значения пользователем

\* @remarks При несовпадении типов возращает код ошибки /c EXIT\_FAILURE

\*/

int input(void);

/\*\*

\* @brief Функция проверки введенного значени

\* @remarks если значение 0 и меньше возвращает код ошибки /c EXIT\_FAILURE

\*/

size\_t get\_above\_zero\_value(void);

/\*\*

\* @brief Функция выполнения задания 1

\* @param arr - массив, задаваемый пользователем

\* @param size\_array - размер массива

\* @return Возвращает сумму чисел согласно условию

\*/

int task1(const int\* arr, const size\_t size\_array);

/\*\*

\* @brief Функция выполнения задания 2

\* @param arr - массив, задаваемый пользователем

\* @param size\_array - размер массива

\* @remarks Выводит индексы согласно условию

\*/

void task2(const int\* arr, const size\_t size\_array);

/\*\*

\* @brief Функция выполнения задания 3

\* @param arr - массив, задаваемый пользователем

\* @param size\_array - размер массива

\* @return Возвращает измененный массив

\*/

void task3(int\* arr, const size\_t size\_array);

/\*\*

\* @brief Функция проверки меньшего и большего значений интервала выбора чисел для рандомного заполнения массива

\* @param low\_value - меньшее значение

\* @param high\_value - большее значение

\* @remarks при непрохождении проверки возвращает код ошибки /c EXIT\_FAILURE

\*/

void check\_interval(const int low\_value, const int high\_value);

/\*\*

\* @brief Функция проверки массива

\* @param arr - массив, который мы проверяем на NULL

\* @remarks при непрохождении проверки возвращает код ошибки /c EXIT\_FAILURE

\*/

void check\_array(int\* const arr);

/\*\*

\* @brief Функция перекопирования массива

\* @param arr - изачальный массив

\* @param size\_array - размер массивов

\* @return Возвращает массив со значениями изначального массива

\*/

int\* copy\_array(int\* const arr, const size\_t size\_array);

/\*\*

\* @brief Функция создания массива

\* @param size\_array - размер создаваемого массива

\* @return возвращает созданный по размеру массив

\*/

int\* create\_array(size\_t const size\_array);

/\*\*

\* @brief Функция проверки наличия 3-го элемента для задания 3

\* @remarks Завершает программу с кодом ошибки /c EXIT\_FAILURE если в массиве меньше 3-х элементов

\*/

void check\_third\_element(const size\_t size\_array);

/\*\*

\* @brief Выбор исполняемой функции

\* @param fill\_random - заполнение случайными числами

\* @param fill\_input - заполнение числами пользователя

\*/

enum Task

{

fill\_random = 1,

fill\_input,

};

/\*\*

\* @brief Точка входа

\* @return Возвращает код ошибки 0

\*/

int main(void)

{

srand(time(NULL));

puts("Enter command,size of array, low and high value of array");

int command = input();

size\_t size\_array = get\_above\_zero\_value();

int low\_value = input(), high\_value = input();

check\_interval(low\_value, high\_value);

int\* arr = create\_array(size\_array);

fill\_array(command, arr, low\_value, high\_value, size\_array);

print\_array(arr, size\_array);

int\* arr\_ = copy\_array(arr, size\_array);

printf("task1: %d\n", task1(arr\_, size\_array));

puts("task2: ");

task2(arr\_, size\_array);

puts("task3: ");

task3(arr\_, size\_array);

print\_array(arr\_, size\_array);

free(arr\_);

free(arr);

return 0;

}

int\* create\_array(size\_t const size\_array)

{

int\* arr = (int\*)malloc(size\_array \* sizeof(int));

check\_array(arr);

return arr;

}

int\* copy\_array(int\* const arr, const size\_t size\_array)

{

int\* arr\_ = create\_array(size\_array);

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

{

arr\_[i] = arr[i];

}

return arr\_;

}

void check\_array(int\* const arr)

{

if (arr == NULL)

{

printf("Memory input");

exit(EXIT\_FAILURE);

}

}

void check\_interval(const int low\_value, const int high\_value)

{

if (low\_value > high\_value)

{

puts("Interval error");

exit(EXIT\_FAILURE);

}

}

void fill\_array(const int command, int\* array, const int low\_value, const int high\_value, const size\_t size\_array)

{

switch ((enum Task)command)

{

case fill\_random: fill\_by\_random(array, low\_value, high\_value, size\_array); break;

case fill\_input: fill\_by\_input(array, size\_array); break;

default: puts("Command error"); exit(EXIT\_FAILURE);

}

}

void fill\_by\_input(int\* array, const size\_t size\_array)

{

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

{

array[i] = input();

}

}

void fill\_by\_random(int\* array, const int low\_value, const int high\_value, const size\_t size\_array)

{

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

{

array[i] = rand() % (high\_value - low\_value + 1) + low\_value;

}

}

int input(void)

{

int input = 0;

if (scanf("%d", &input) != 1)

{

puts("Input error");

exit(EXIT\_FAILURE);

}

return input;

}

size\_t get\_above\_zero\_value(void)

{

int input\_ = input();

if (input <= 0)

{

puts("Input error");

exit(EXIT\_FAILURE);

}

return (size\_t)input\_;

}

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

{

puts("[");

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

{

printf("%d,", array[i]);

}

puts("]");

}

int task1(const int\* arr, const size\_t size\_array)

{

int s = 0;

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

{

if (abs(arr[i]) < 10)

{

s += arr[i];

}

}

return s;

}

void task2(const int\* arr, const size\_t size\_array)

{

for (size\_t i = 0; i < size\_array - 1; i++)

{

if ((arr[i] > arr[i + 1]))

{

printf("%d\n", i);

}

}

}

void task3(int\* arr, const size\_t size\_array)

{

check\_third\_element(size\_array);

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

{

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

{

arr[i] \*= arr[2];

}

}

}

void check\_third\_element(const size\_t size\_array)

{

if (size\_array < 2)

{

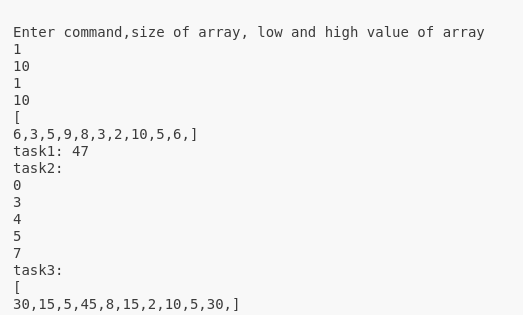
printf("Task 3 cannot be completed");

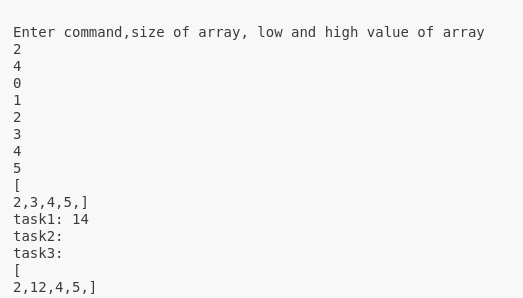
exit(EXIT\_FAILURE);

}

}

* 1. Решение тестового примера

Рисунок 16 — решение тестового примера рандомным заполнением

Рисунок 17 — решение тестового примера заполнением пользователем

* 1. Зачёт задания в GitHub

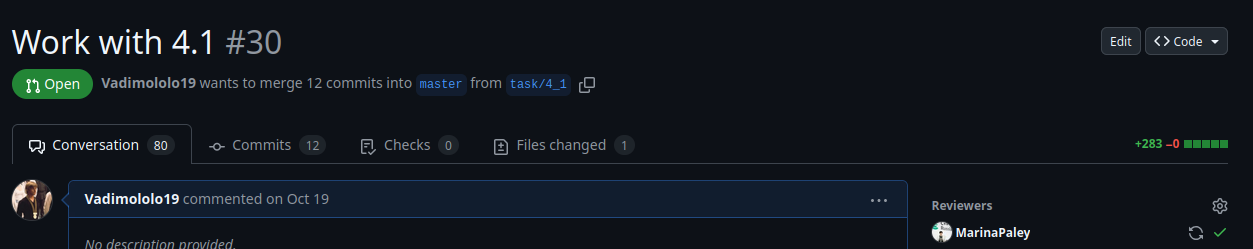


Рисунок 18 – Зачёт задания в GitHub

1. Задание 4.2
   1. Формулировка задания

Создать одномерный массив 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. Составить блок-схему.

|  |  |  |
| --- | --- | --- |
| **Вариант** | **Задачи** | **Интервал** |
| **9** | 1. Заменить минимальный по модулю положительный элемент массива нулем. 2. Удалить из него все элементы, первая и последняя цифра которых четная. 3. Из элементов массива D сформировать массив A той же размерности по правилу: элементы с 3-го по 12-й находятся по формуле Ai = -Di2, остальные по формуле Ai = Di-1 | [-40;40] |

* 1. Блок-схема алгоритма

Блок-схема алгоритма представлена на рисунках ( - 34)

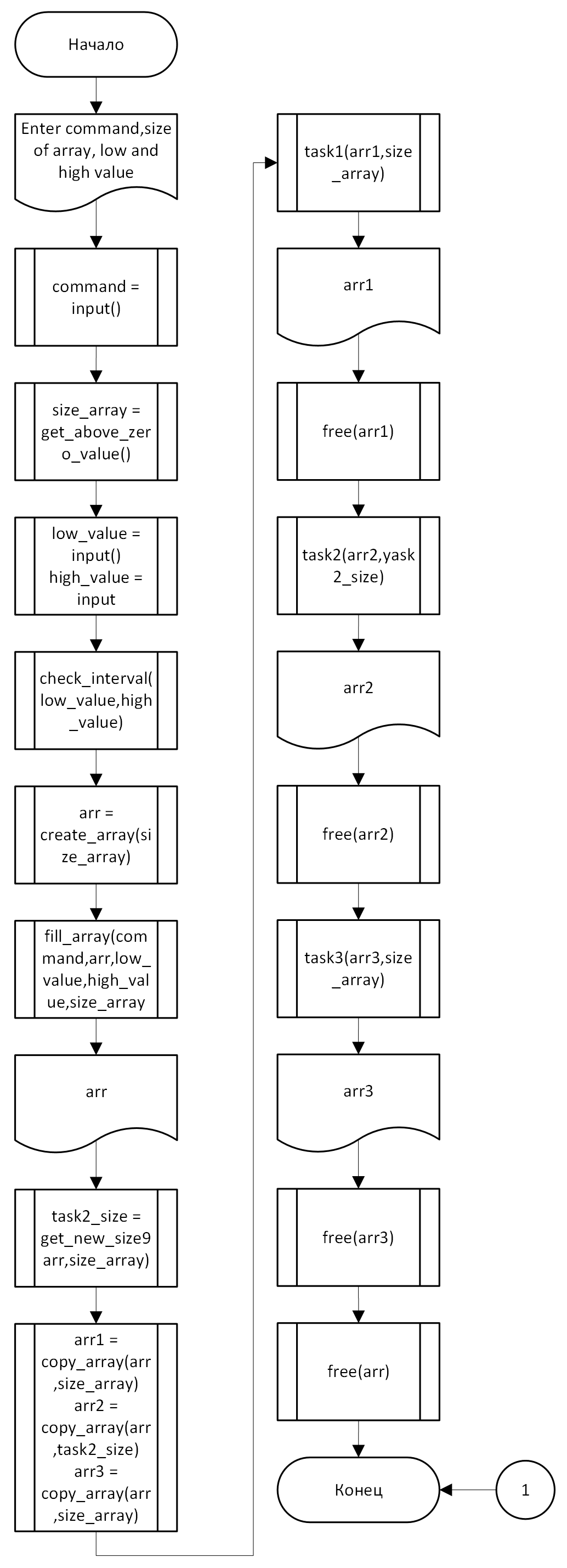


Рисунок 19 – Блок-схема функции main

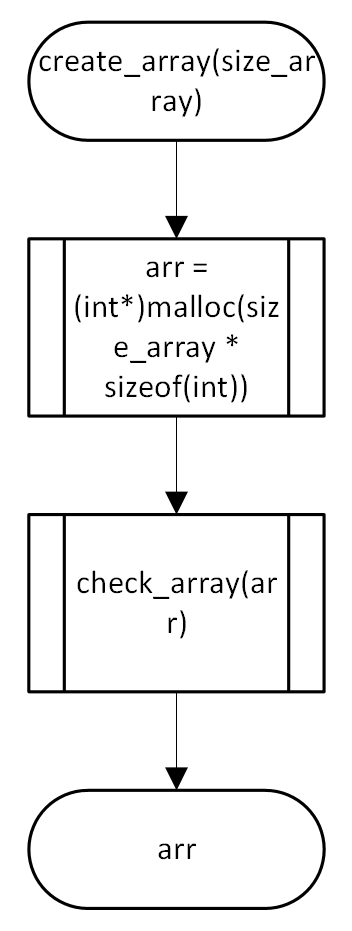


Рисунок 20 – Блок-схема функции create\_array

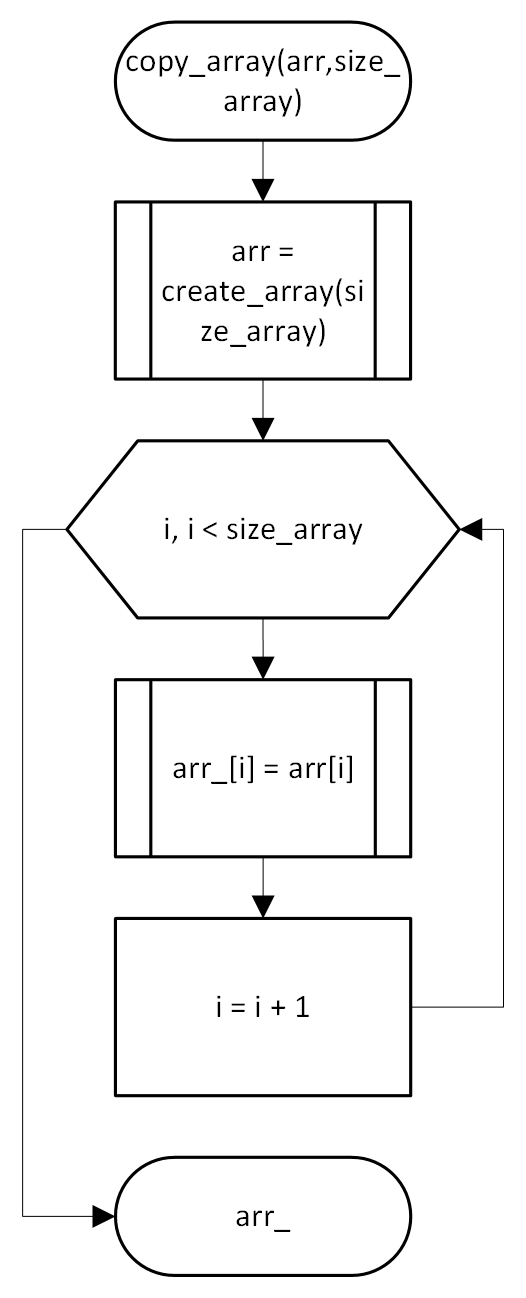


Рисунок 21 – Блок-схема функции copy\_array

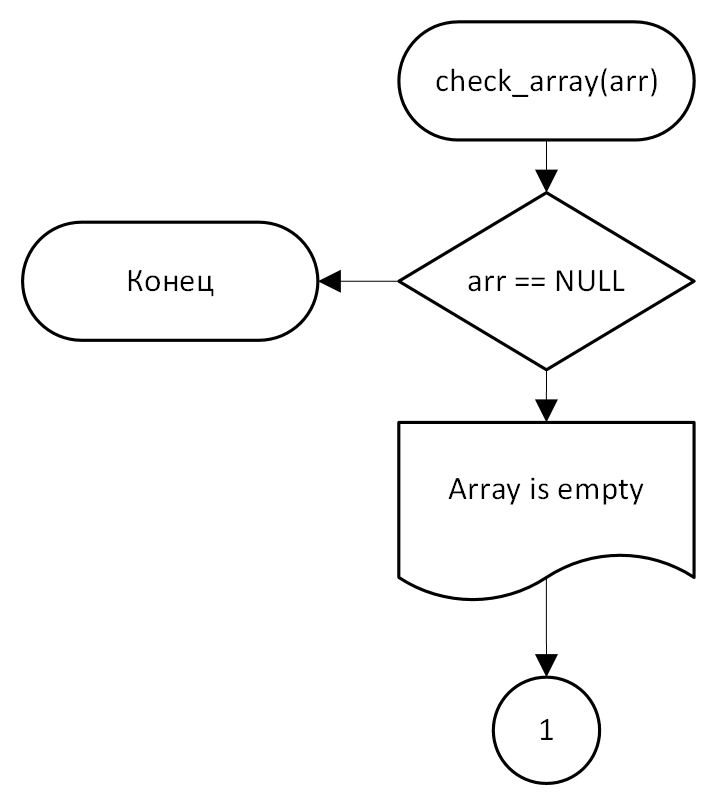


Рисунок 22 – Блок-схема функции check\_array

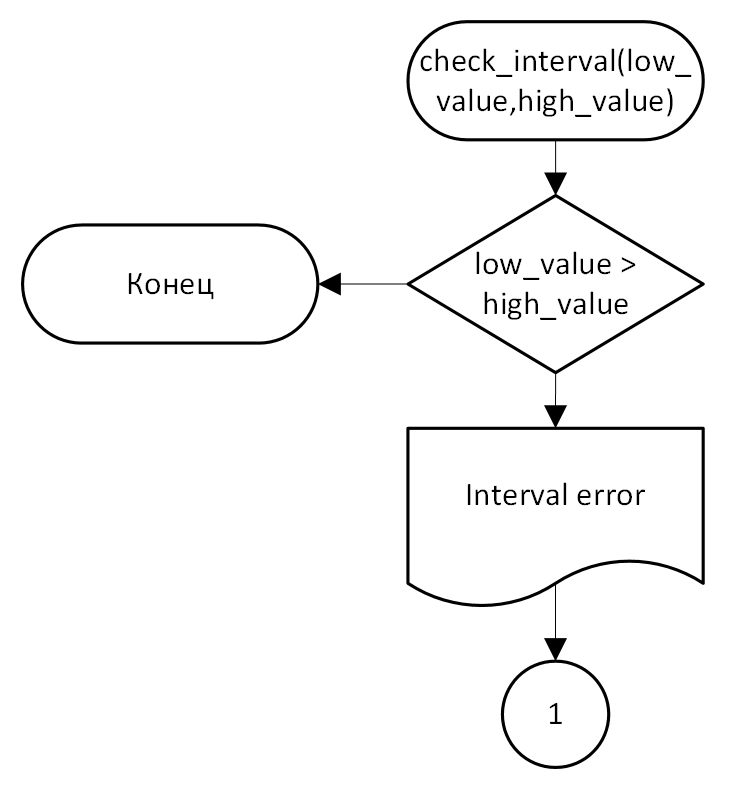
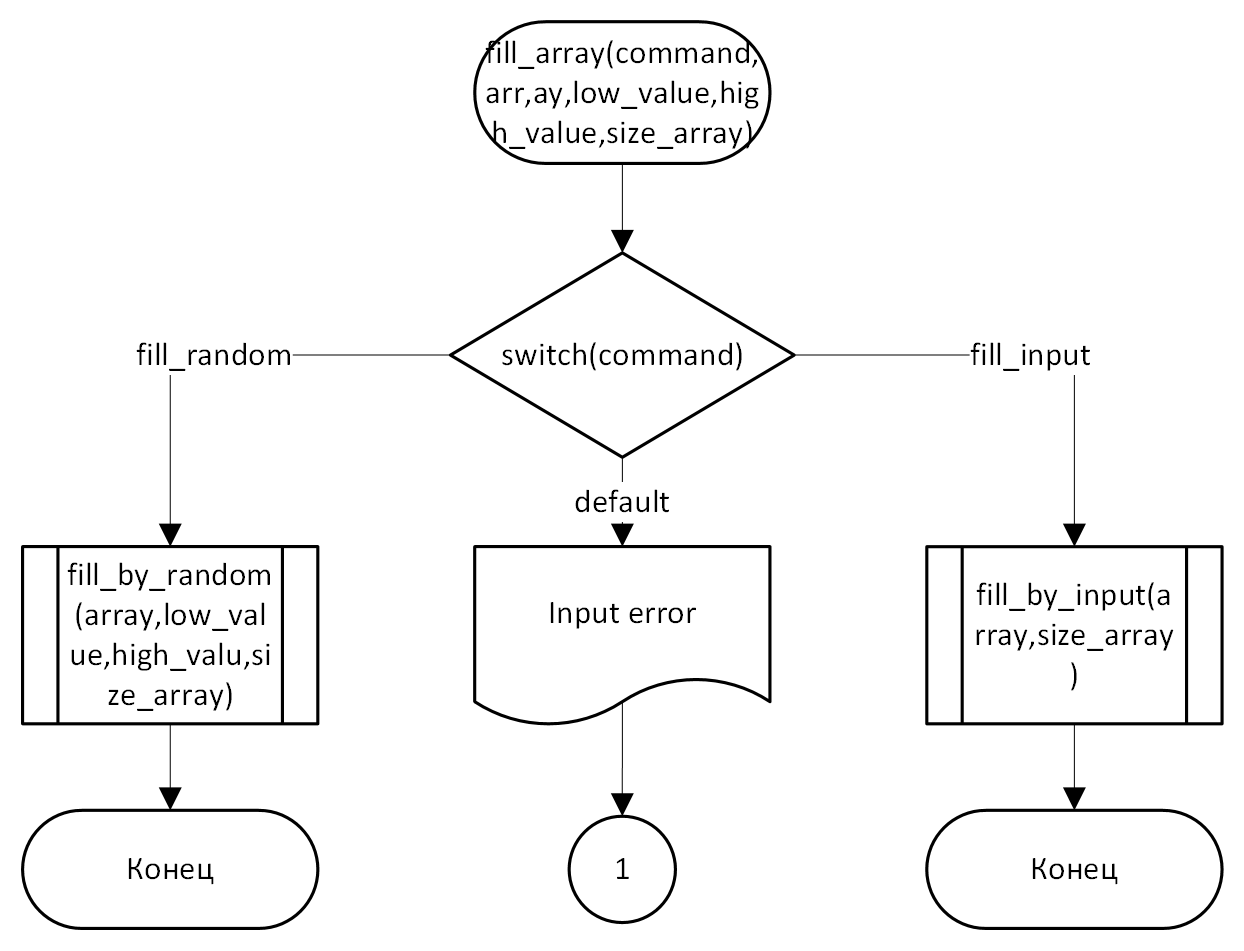


Рисунок 23 – Блок-схема функции check\_interval

Рисунок 24 - Блок-схема функции fill\_array

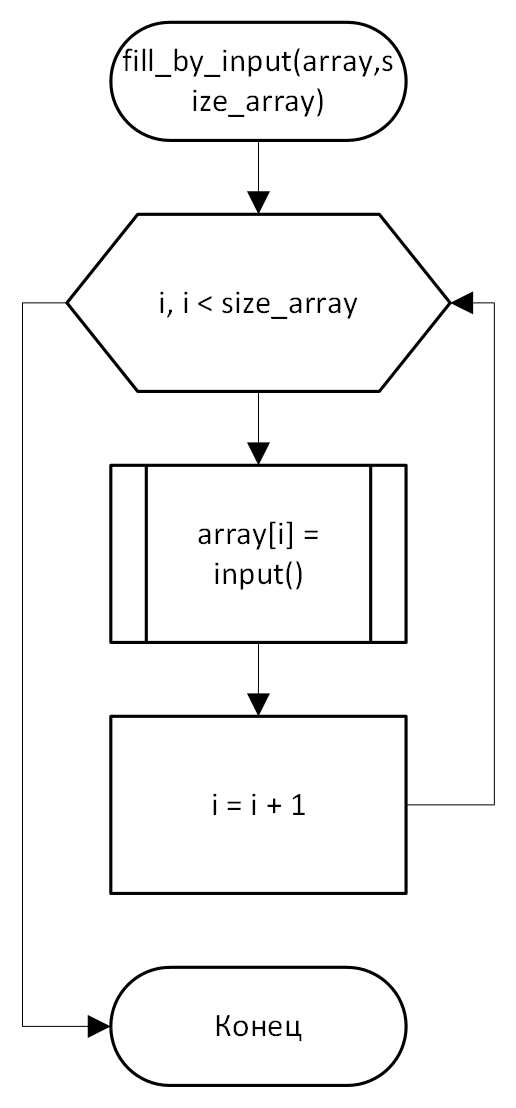


Рисунок 25 – Блок-схема функции fill\_by\_input

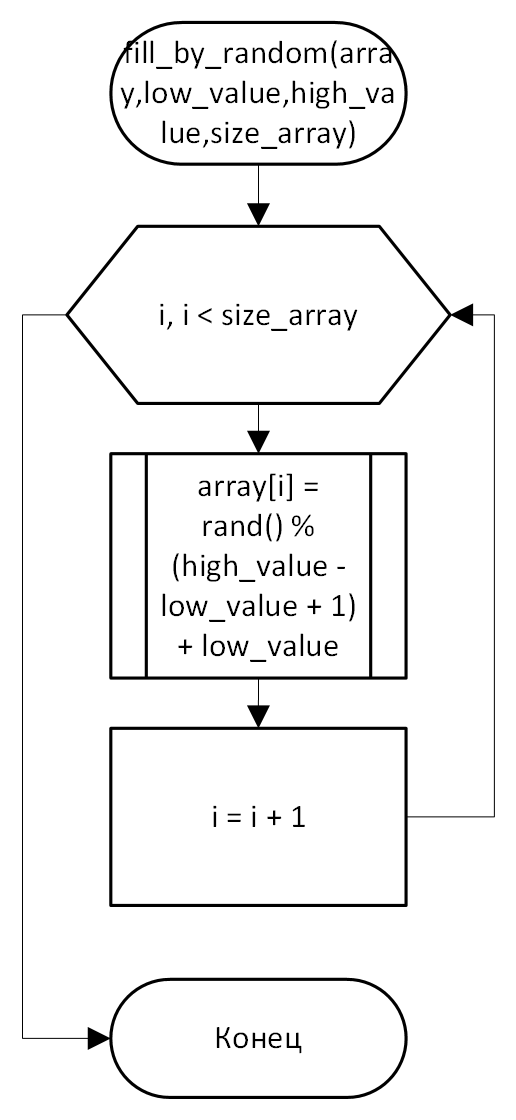


Рисунок 26 – Блок-схема функции fill\_by\_random

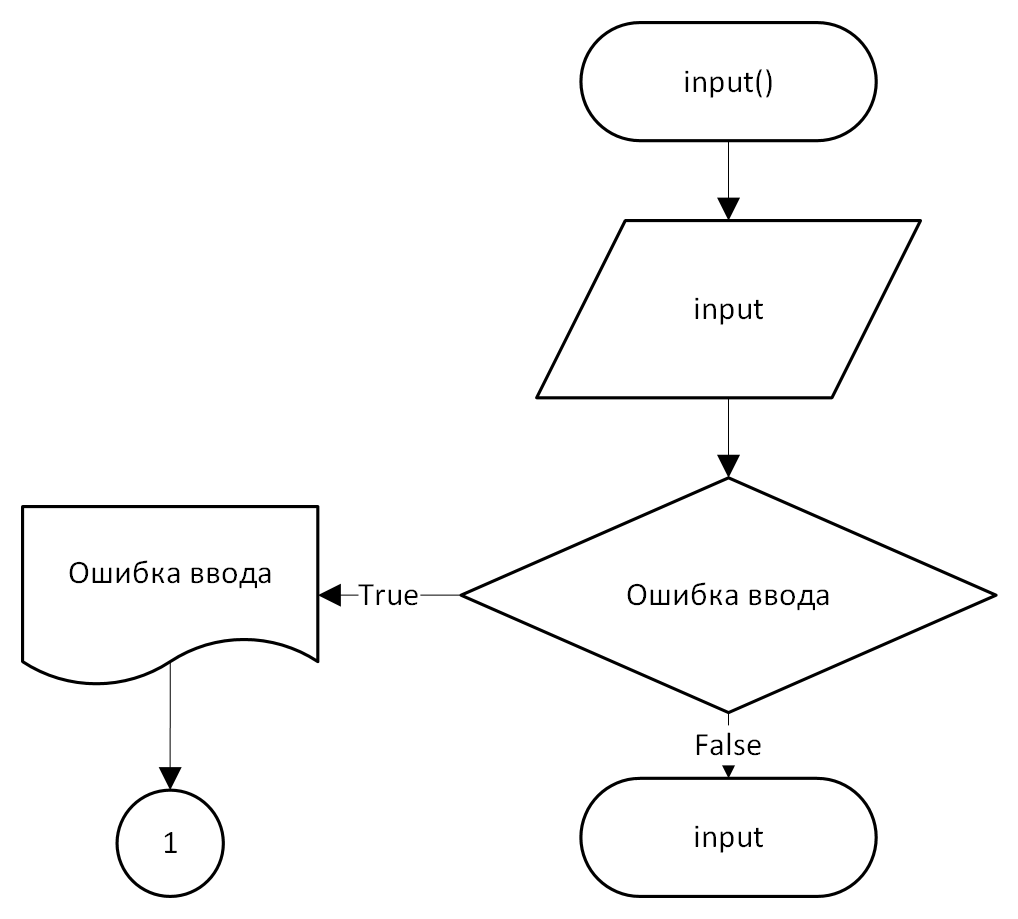


Рисунок 27 – Блок-схема функции input

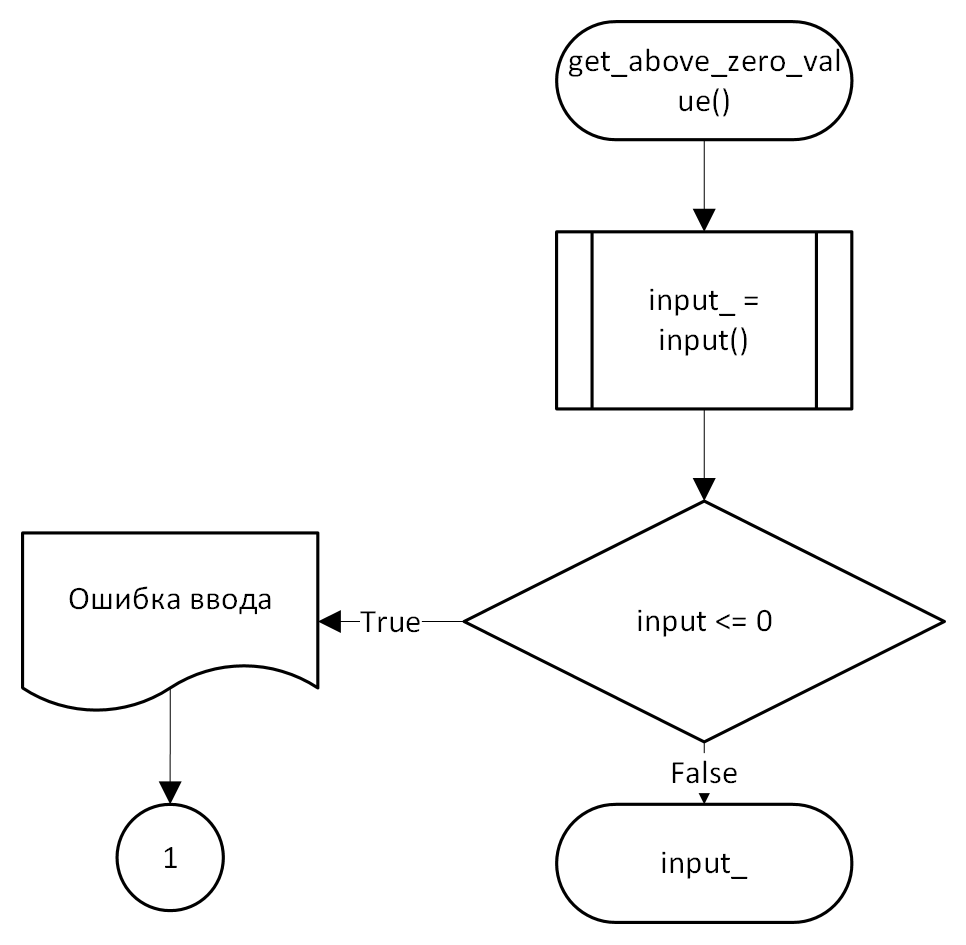


Рисунок 28 – Блок-схема функции get\_above\_zero\_value

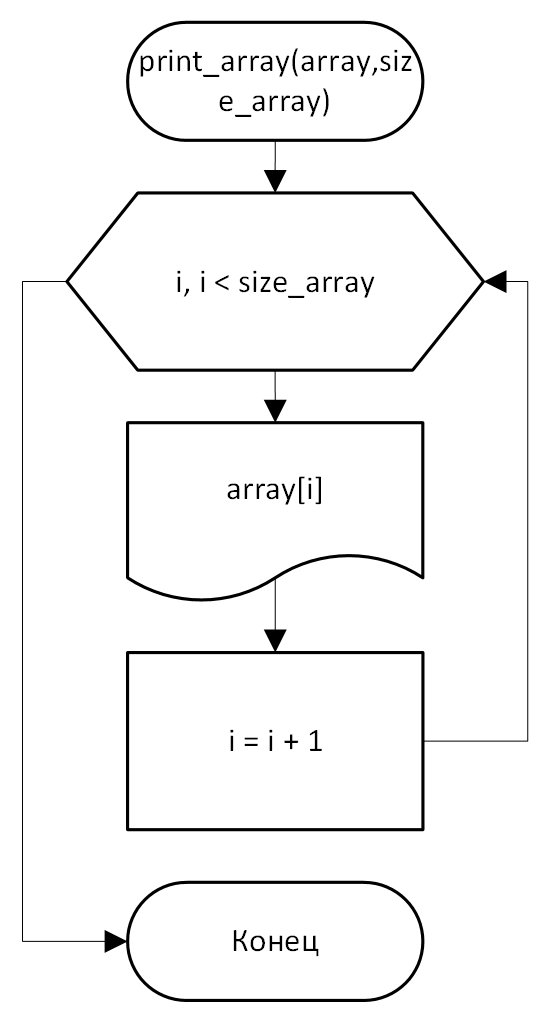


Рисунок 29 – Блок-схема функции print\_array

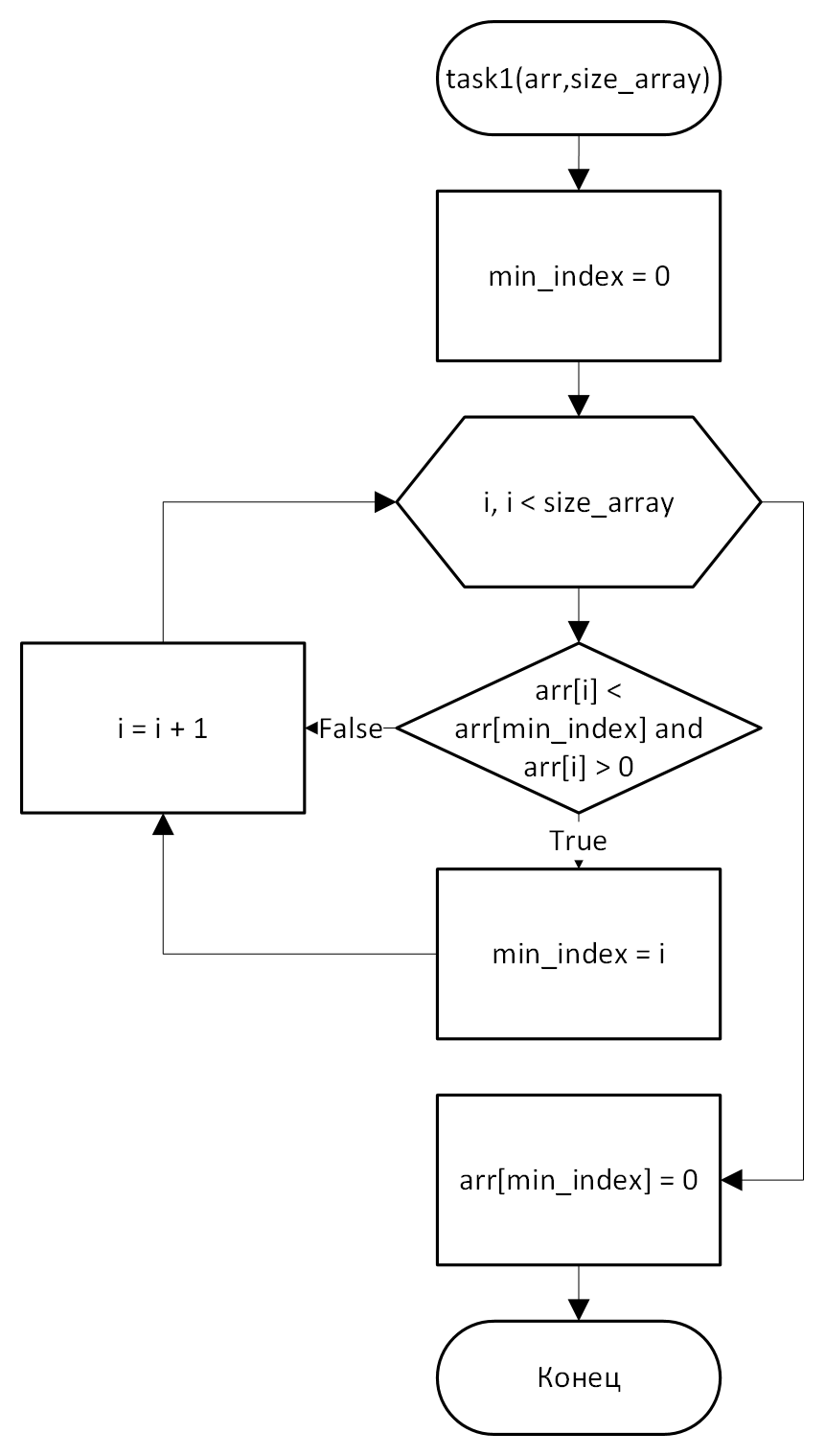


Рисунок 30 – Блок-схема функции task1

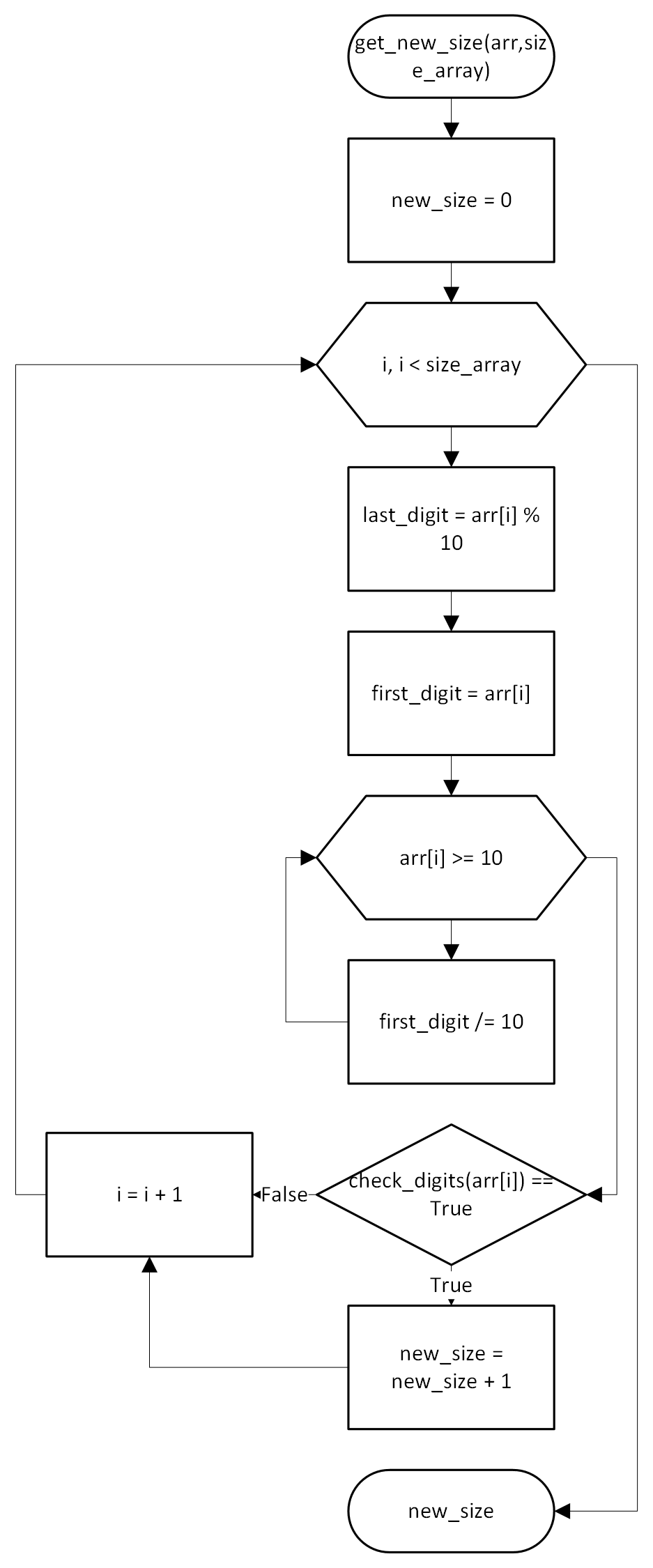


Рисунок 31 – Блок-схема функции get\_new\_size

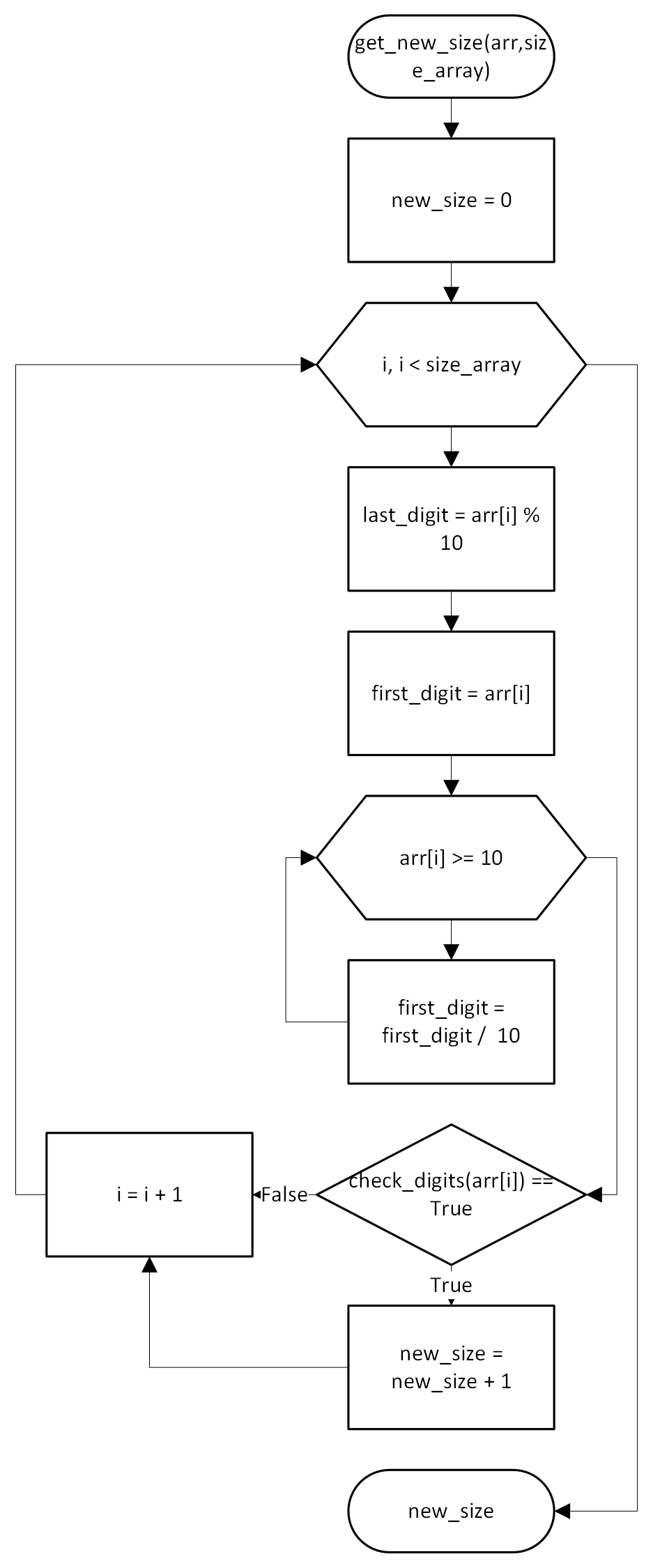
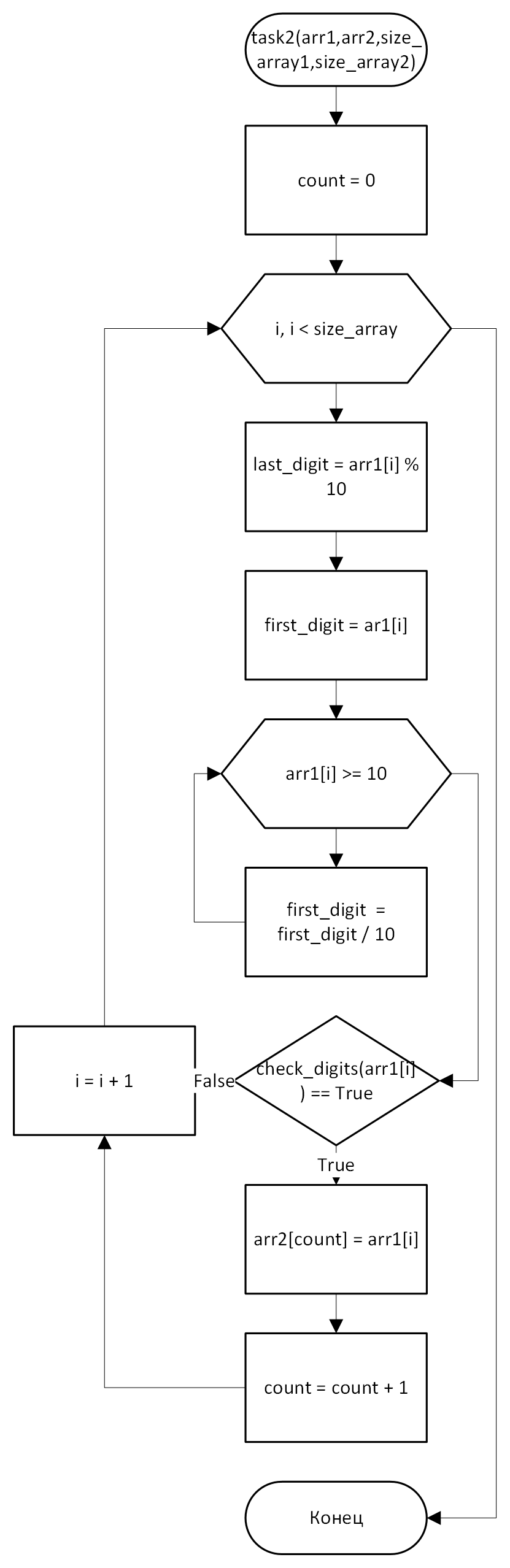
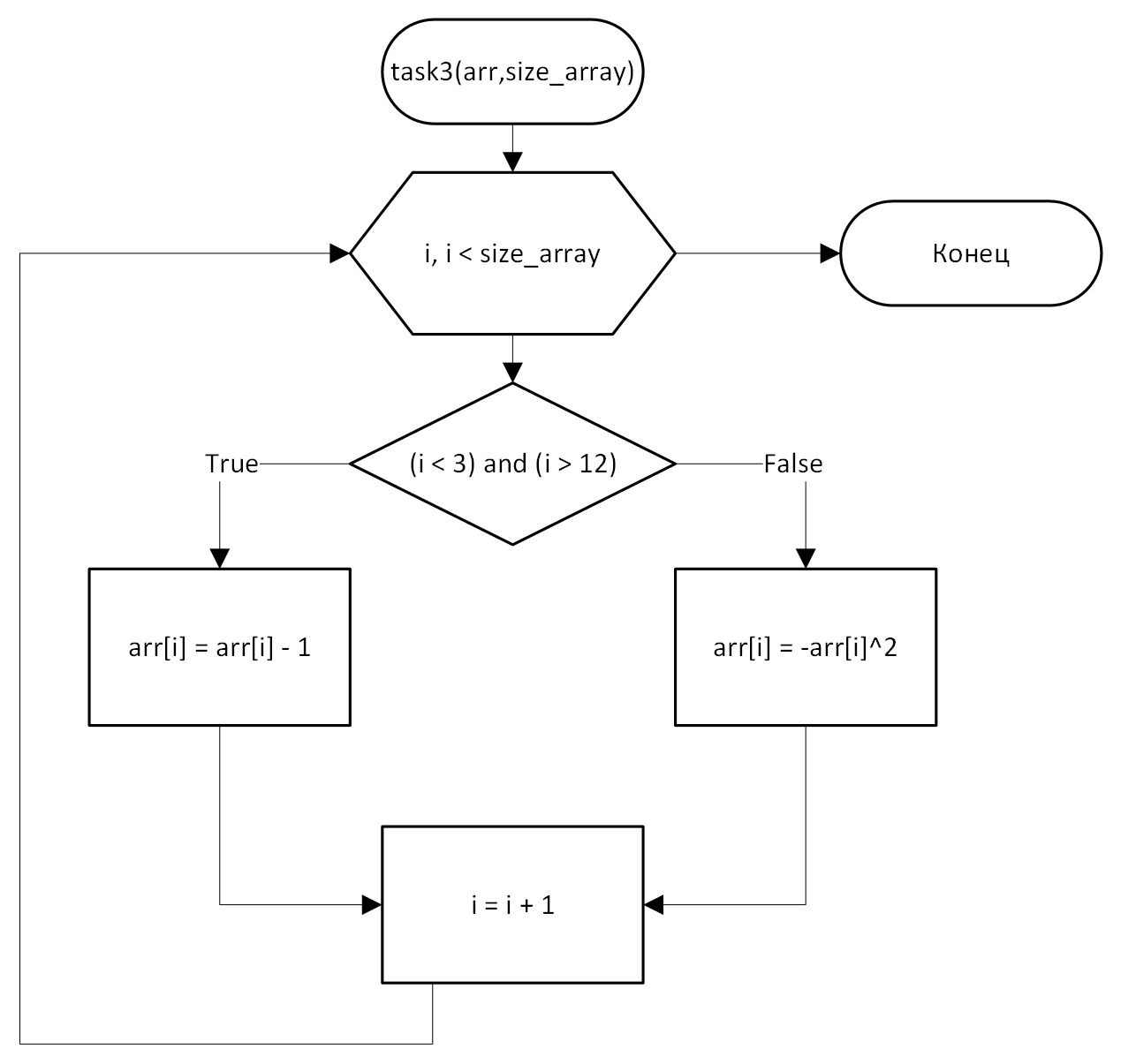


Рисунок 32 – Блок-схема функции check\_digits

Рисунок 33 — Блок-схема функции task2

Рисунок 34 - Блок-схема функции task3

* 1. Код задания 4.2

#include <stdio.h>

#include <stdlib.h>

#include <math.h>

#include <time.h>

#include <limits.h>

#include <stdbool.h>

/\*\*

\* @brief Функция выбора метода заполнения массива пользователем

\* @param command - переменная для выбора метода заполнения массива

\* @param array - массив, задаваемый пользователем

\* @param low\_value - начало массива

\* @param size\_array - размер массива

\* @remarks при неправильном вводе команды выводит ошибку

\*/

void fill\_array(const int command, int\* array, const int low\_value, const int high\_value, const size\_t size\_array);

/\*\*

\* @brief Функция заполнения массива интервалом, который задал пользователь

\* @param array - массив, задаваемый пользователем

\* @param size\_array - размер массива

\*/

void fill\_by\_input(int\* array, const size\_t size\_array);

/\*\*

\* @brief Функция заполнения массива случайными числами

\* @param array - массив, задаваемый пользователем

\* @param low\_value - начало массива

\* @param size\_array - размер массива

\*/

void fill\_by\_random(int\* array, const int low\_value, const int high\_value, const size\_t size\_array);

/\*\*

\* @brief Функция вывода массива

\* @param array - массив, задаваемый пользователем

\* @param size\_array - размер массива

\*/

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

/\*\*

\* @brief Функция ввода значения пользователем

\* @remarks При несовпадении типов возращает код ошибки /c EXIT\_FAILURE

\*/

int input(void);

/\*\*

\* @brief Функция проверки введенного значени

\* @remarks если значение 0 и меньше возвращает код ошибки /c EXIT\_FAILURE

\*/

size\_t get\_above\_zero\_value(void);

/\*\*

\* @brief Функция проверки меньшего и большего значений интервала выбора чисел для рандомного заполнения массива

\* @param low\_value - меньшее значение

\* @param high\_value - большее значение

\* @remarks при непрохождении проверки возвращает код ошибки /c EXIT\_FAILURE

\*/

void check\_interval(const int low\_value, const int high\_value);

/\*\*

\* @brief Функция проверки массива

\* @param arr - массив, который мы проверяем на NULL

\* @remarks при непрохождении проверки возвращает код ошибки /c EXIT\_FAILURE

\*/

void check\_array(int\* const arr);

/\*\*

\* @brief Функция перекопирования массива

\* @param arr - изачальный массив

\* @param size\_array - размер массивов

\* @return Возвращает массив со значениями изначального массива

\*/

int\* copy\_array(int\* const arr, const size\_t size\_array);

/\*\*

\* @brief Функция создания массива

\* @param size\_array - размер создаваемого массива

\* @return возвращает созданный по размеру массив

\*/

int\* create\_array(size\_t const size\_array);

/\*\*

\* @brief Функция выполнения первого задания

\* @param arr - массив который мы изменяем

\* @param size\_arrray - размер нашего массива

\* @remarks Заменяет минимальный положительный элемент по модулю

\*/

void task1(int\* arr, const size\_t size\_array);

/\*\*

\* @brief Функция выполнения второго задания

\* @param arr1 - изначальный массив

\* @param size\_array1 - размер изначального массива

\* @param arr2 - заполняемый массив

\* @paran size\_array2 - размер нового массива

\* @return Возвращает новый массив по заданию

\*/

void task2(const int\* arr1, int\* arr2,const size\_t size\_array1, size\_t size\_array2);

/\*\*

\* @brief Функция выполнения третьего задания

\* @param arr - массив, который мы изменяем

\* @param size\_array - размер нашего массива

\* @remarks Из элементов массива D сформировать массив A той же размерности по правилу: элементы с 3-го по 12-й находятся по формуле Ai = -Di2, остальные по формуле Ai = Di-1

\*/

void task3(int\* arr, const size\_t size\_array);

/\*\*

\* @brief Функция получения нового размера для задания 2

\* @param arr - изначальный массив

\* @param size\_array - размер изначального массива

\* @return возвращает размер нового массива для задания 2

\*/

size\_t get\_new\_size(const int\* arr, const size\_t size\_array);

/\*\*

\* @brief Функция првоверки первой и последней цифры в числе

\* @param value - число, цифра которой мы проверяем

\* @return возвращает True или False в зависимости от результатов проверки

\*/

bool check\_digits(const int value);

/\*\*

\* @brief Выбор исполняемой функции

\* @param fill\_random - заполнение случайными числами

\* @param fill\_input - заполнение числами пользователя

\*/

enum Task

{

fill\_random = 1,

fill\_input,

};

/\*\*

\* @brief Точка входа

\* @return Возвращает код ошибки 0

\*/

int main(void)

{

srand(time(NULL));

puts("Enter command,size of array, low and high value of array");

int command = input();

size\_t size\_array = get\_above\_zero\_value();

int low\_value = input(), high\_value = input();

check\_interval(low\_value, high\_value);

int\* arr = create\_array(size\_array);

fill\_array(command, arr, low\_value, high\_value, size\_array);

print\_array(arr, size\_array);

size\_t task2\_size = get\_new\_size(arr, size\_array);

int\* arr1 = copy\_array(arr, size\_array),\*arr2 = copy\_array(arr,task2\_size), \*arr3 = copy\_array(arr,size\_array);

task1(arr1, size\_array);

print\_array(arr1, size\_array);

free(arr1);

task2(arr, arr2,size\_array,task2\_size);

print\_array(arr2, task2\_size);

free(arr2);

task3(arr3, size\_array);

print\_array(arr3, size\_array);

free(arr3);

free(arr);

return 0;

}

int\* create\_array(size\_t const size\_array)

{

int\* arr = (int\*)malloc(size\_array \* sizeof(int));

check\_array(arr);

return arr;

}

int\* copy\_array(int\* const arr, const size\_t size\_array)

{

int\* arr\_ = create\_array(size\_array);

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

{

arr\_[i] = arr[i];

}

return arr\_;

}

void check\_array(int\* const arr)

{

if (arr == NULL)

{

printf("Memory input");

exit(EXIT\_FAILURE);

}

}

void check\_interval(const int low\_value, const int high\_value)

{

if (low\_value > high\_value)

{

puts("Interval error");

exit(EXIT\_FAILURE);

}

}

void fill\_array(const int command, int\* array, const int low\_value, const int high\_value, const size\_t size\_array)

{

switch ((enum Task)command)

{

case fill\_random: fill\_by\_random(array, low\_value, high\_value, size\_array); break;

case fill\_input: fill\_by\_input(array, size\_array); break;

default: puts("Command error"); exit(EXIT\_FAILURE);

}

}

void fill\_by\_input(int\* array, const size\_t size\_array)

{

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

{

array[i] = input();

}

}

void fill\_by\_random(int\* array, const int low\_value, const int high\_value, const size\_t size\_array)

{

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

{

array[i] = rand() % (high\_value - low\_value + 1) + low\_value;

}

}

int input(void)

{

int input = 0;

if (scanf\_s("%d", &input) != 1)

{

puts("Input error");

exit(EXIT\_FAILURE);

}

return input;

}

size\_t get\_above\_zero\_value(void)

{

int input\_ = input();

if (input <= 0)

{

puts("Input error");

exit(EXIT\_FAILURE);

}

return (size\_t)input\_;

}

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

{

puts("[");

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

{

printf("%d,", array[i]);

}

puts("]");

}

void task1(int\* arr, const size\_t size\_array)

{

int min\_index= 0;

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

{

if ((arr[i] < arr[min\_index]) && (arr[i] > 0))

{

min\_index = i;

}

}

arr[min\_index] = 0;

}

size\_t get\_new\_size(const int\* arr, const size\_t size\_array)

{

size\_t new\_size = 0;

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

{

int last\_digit = arr[i] % 10;

int first\_digit = arr[i];

while (arr[i] >= 10)

{

first\_digit /= 10;

}

if (check\_digits(arr[i]))

{

new\_size += 1;

}

}

return new\_size;

}

bool check\_digits(const int value)

{

int last\_digit = value % 10;

int first\_digit = value;

while (value >= 10)

{

first\_digit /= 10;

}

if ((first\_digit % 2 == 0 && last\_digit % 2 == 0) || (value < 10 && first\_digit % 2 == 0))

{

return true;

}

return false;

}

void task2(const int\* arr1, int\* arr2, size\_t size\_array1, size\_t size\_array2)

{

int count = 0;

for (size\_t i = 0; i < size\_array1; i++)

{

int last\_digit = arr1[i] % 10;

int first\_digit = arr1[i];

while (arr1[i] >= 10)

{

first\_digit /= 10;

}

if (check\_digits(arr1[i]))

{

arr2[count] = arr1[i];

count++;

}

}

}

void task3(int\* arr, const size\_t size\_array)

{

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

{

if ((i < 3) && (i > 12))

{

arr[i] = arr[i] - 1;

}

else

{

arr[i] = -pow(arr[i], 2);

}}}

* 1. Решение тестового примера

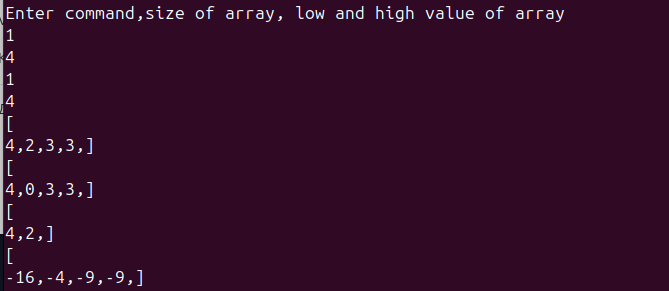


Рисунок 35 – Решение тестового примера рандомным заполнением массива



Рисунок 36 – Решение тестового примера пользовательским заполнением массива

* 1. Зачёт задания в GitHub

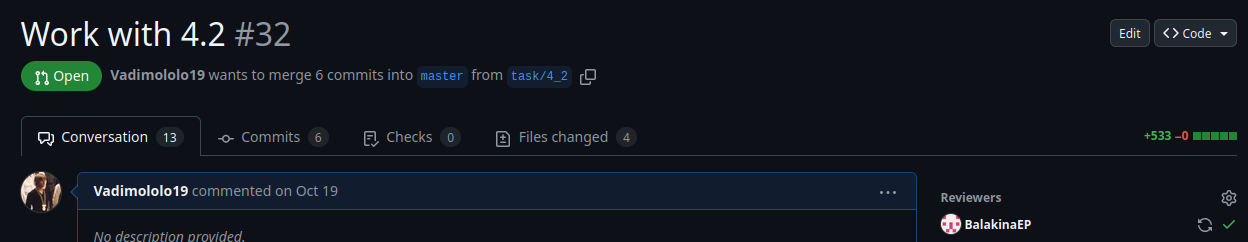


Рисунок 37 – Зачёт задания в GitHub

### 3. Задание 4.3

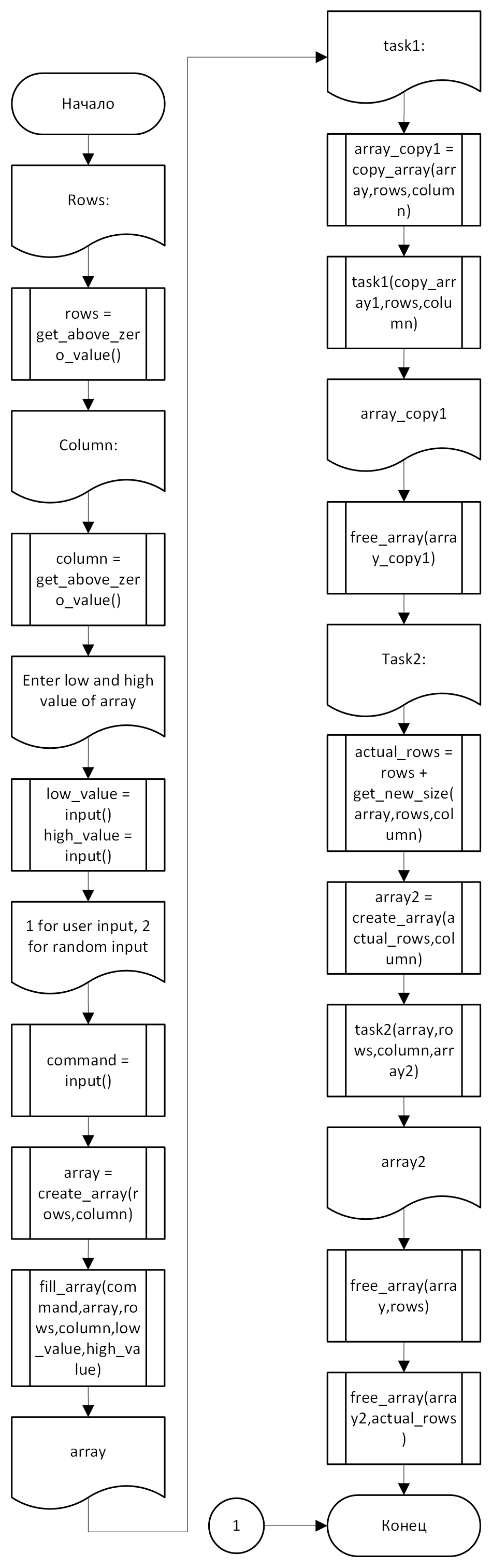
## **3.1. Формулировка задания**

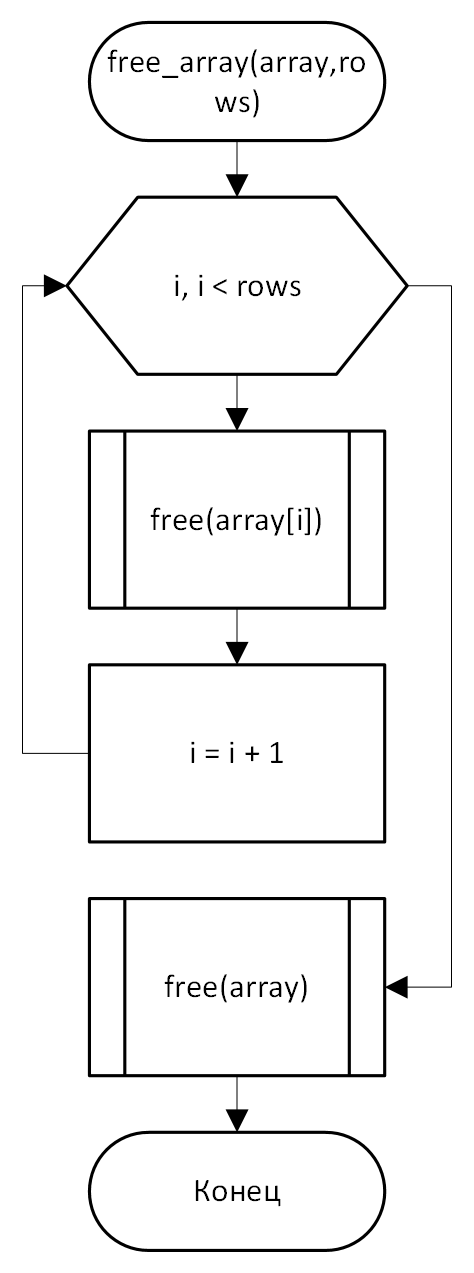
Создать многомерный массив nˣm из *n* целых чисел. Реализовать возможность заполнения массива, как случайными числами, так и с помощью клавиатуры по желанию пользователя. Пункты задания организовать в виде функций (методов). Вывести массив на экран.

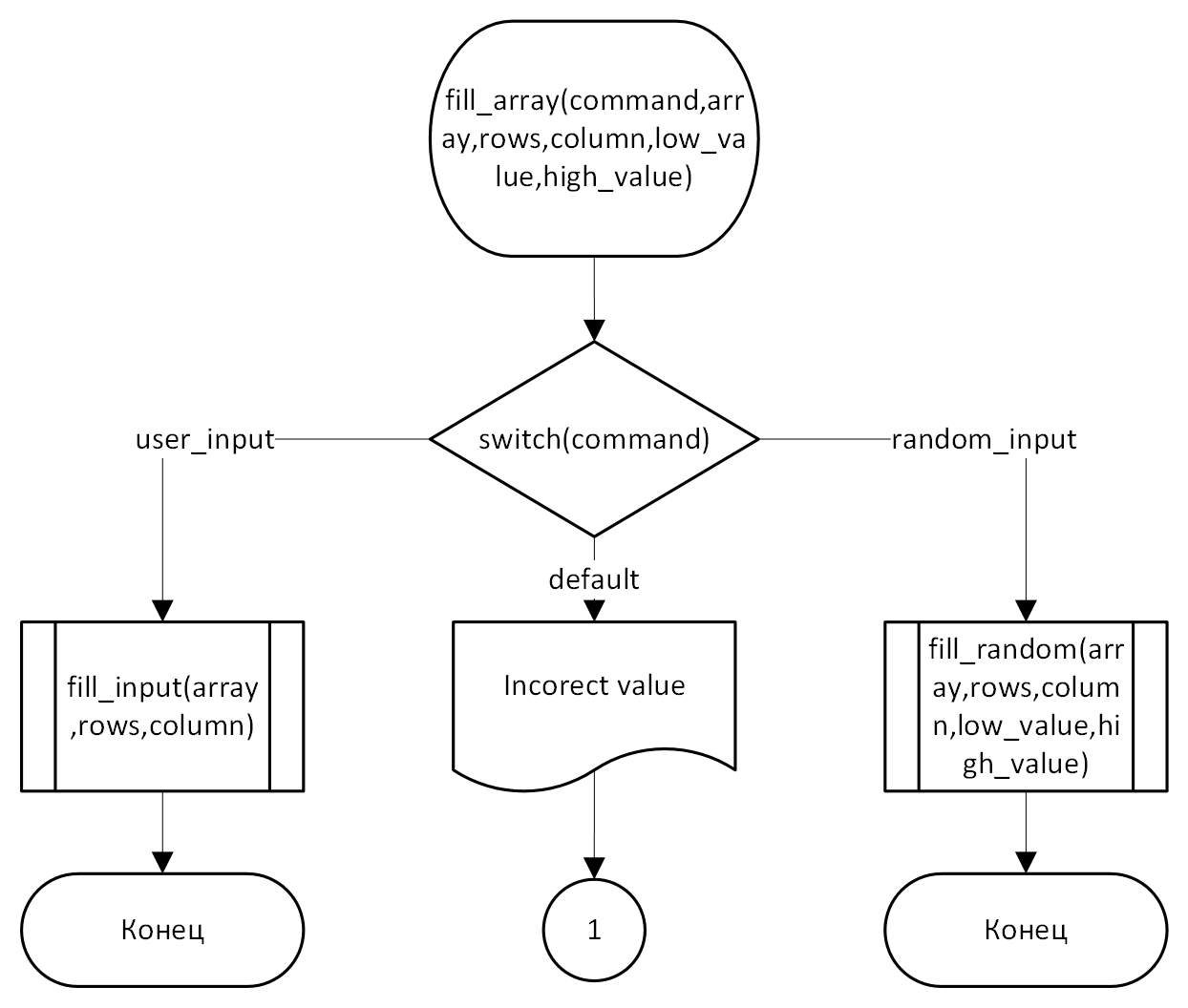
|  |  |
| --- | --- |
| **Вариант** | **Задачи** |
| **9** | 1. Заменить минимальный элемент каждого столбца нулем. 2. Вставить после каждой строки, содержащей максимальный по модулю элемент, последнюю строку. |

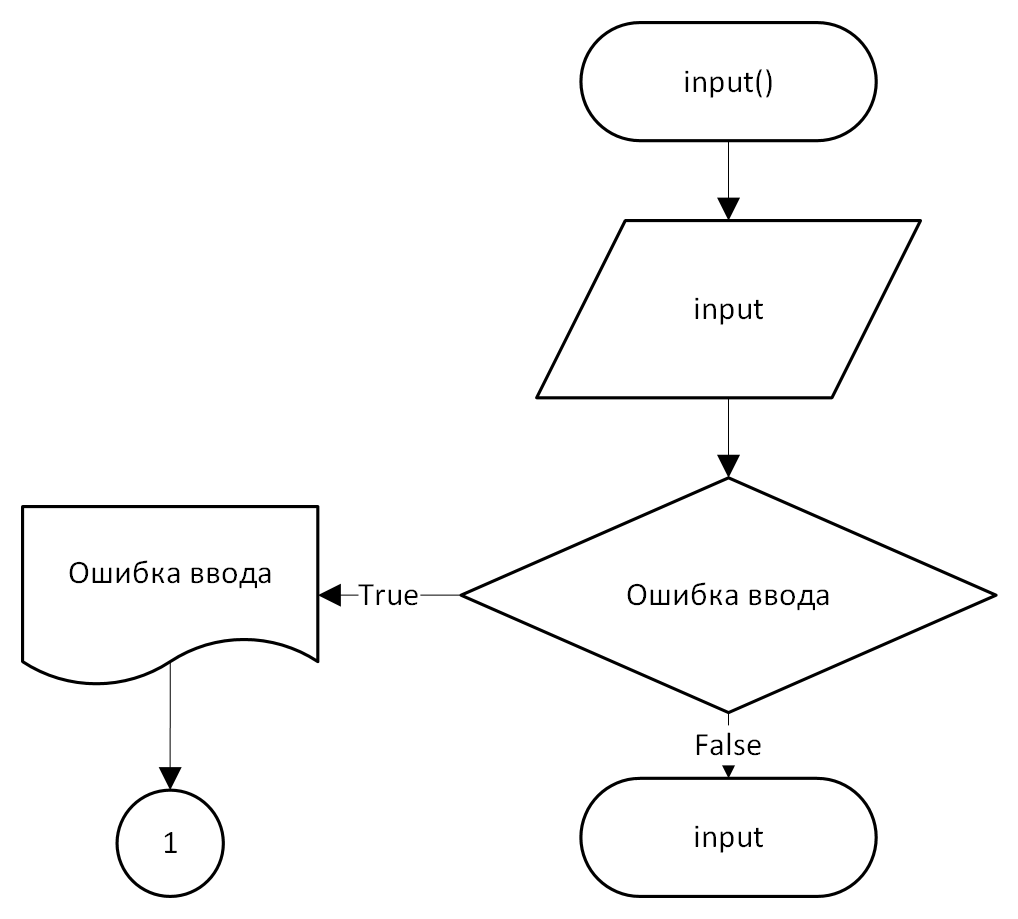
## **3.2 Блок-схема алгоритма**

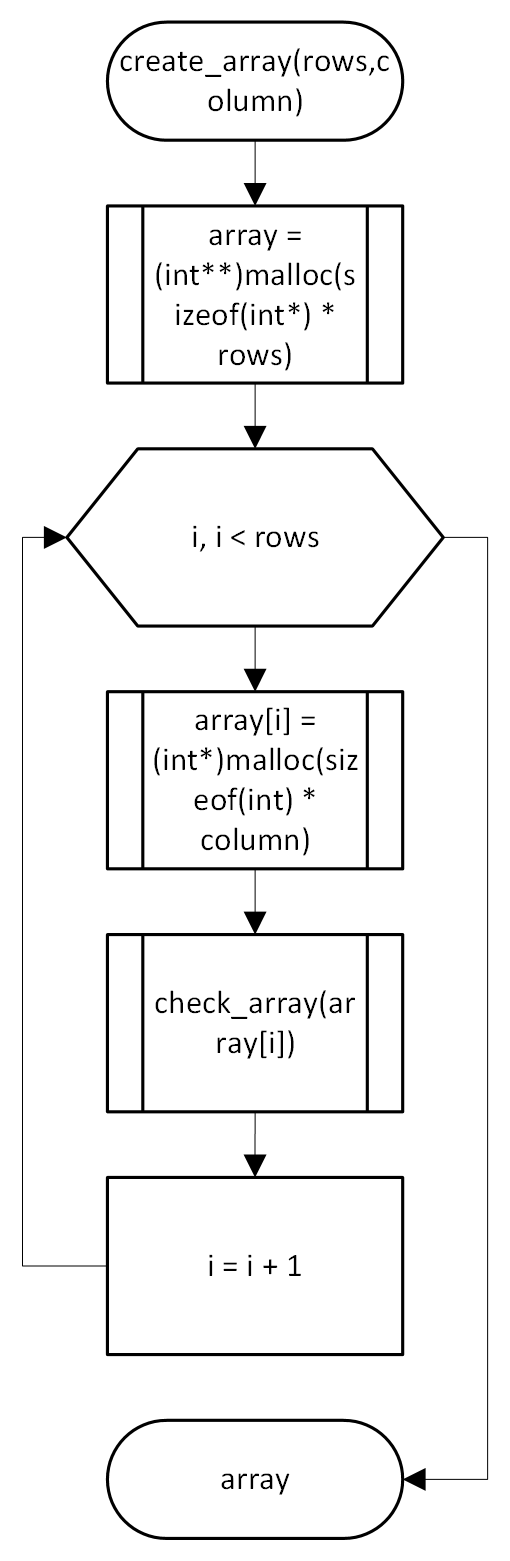
Блок-схема алгоритма представлена на рисунках (Рисунок 38 - 52)

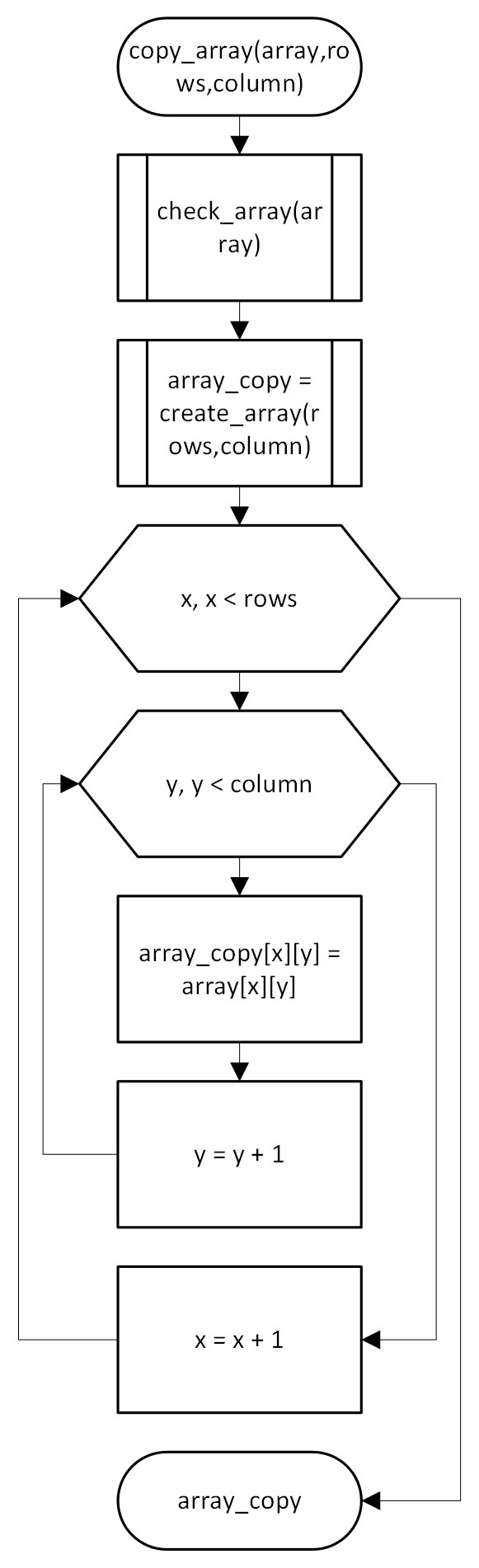
Рисунок 38 - Блок-схема функции main

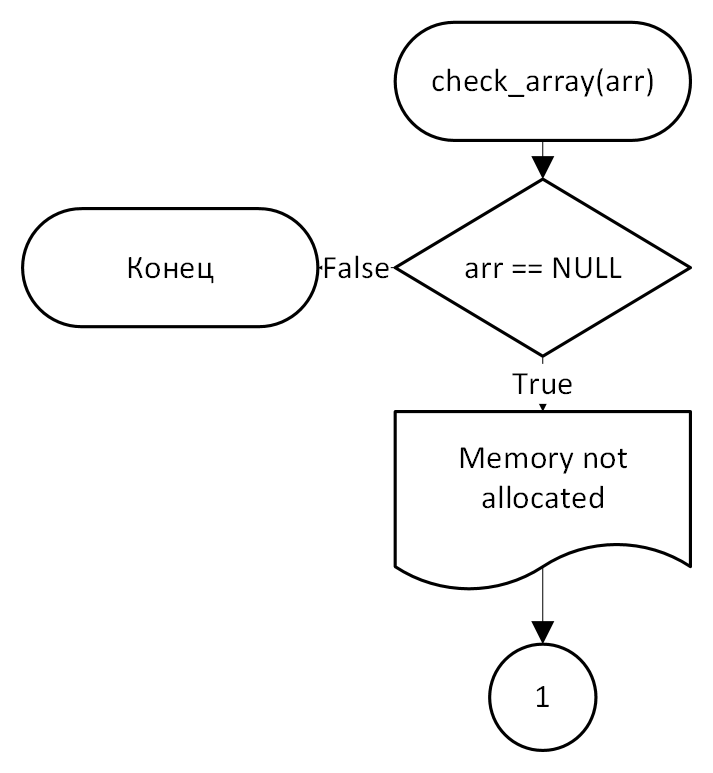
Рисунок 39 - Блок-схема функции free\_array

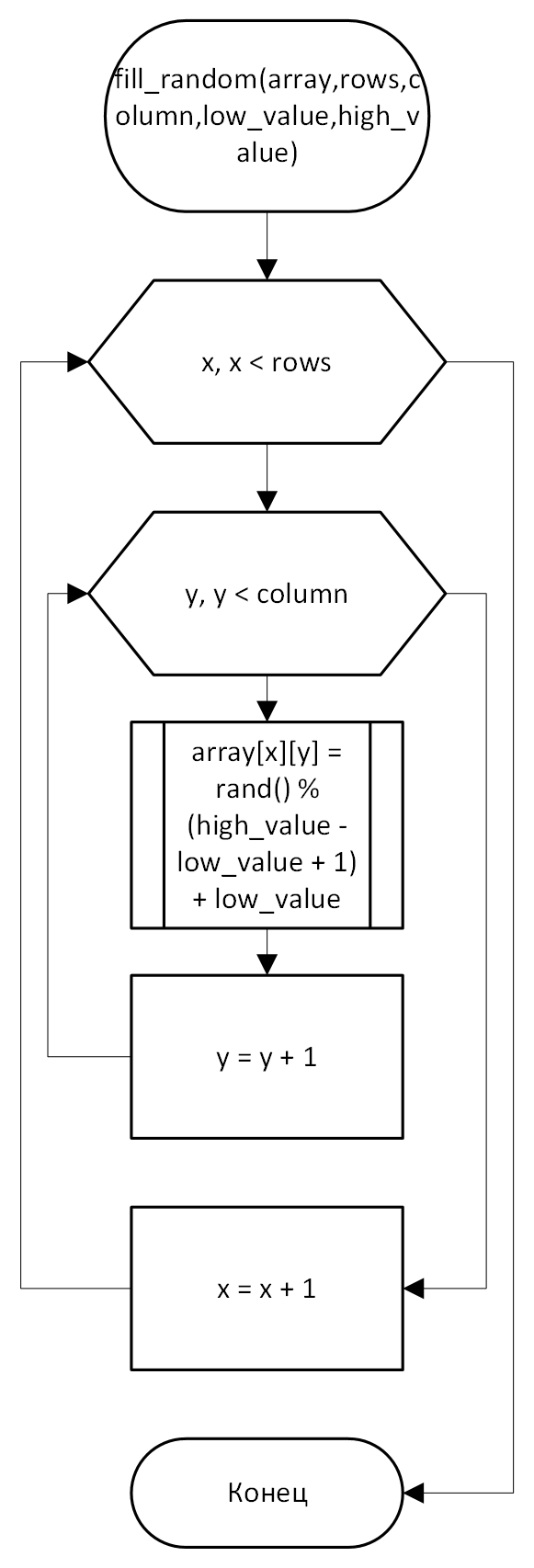
Рисунок 40 — Блок-схема функции fill\_array

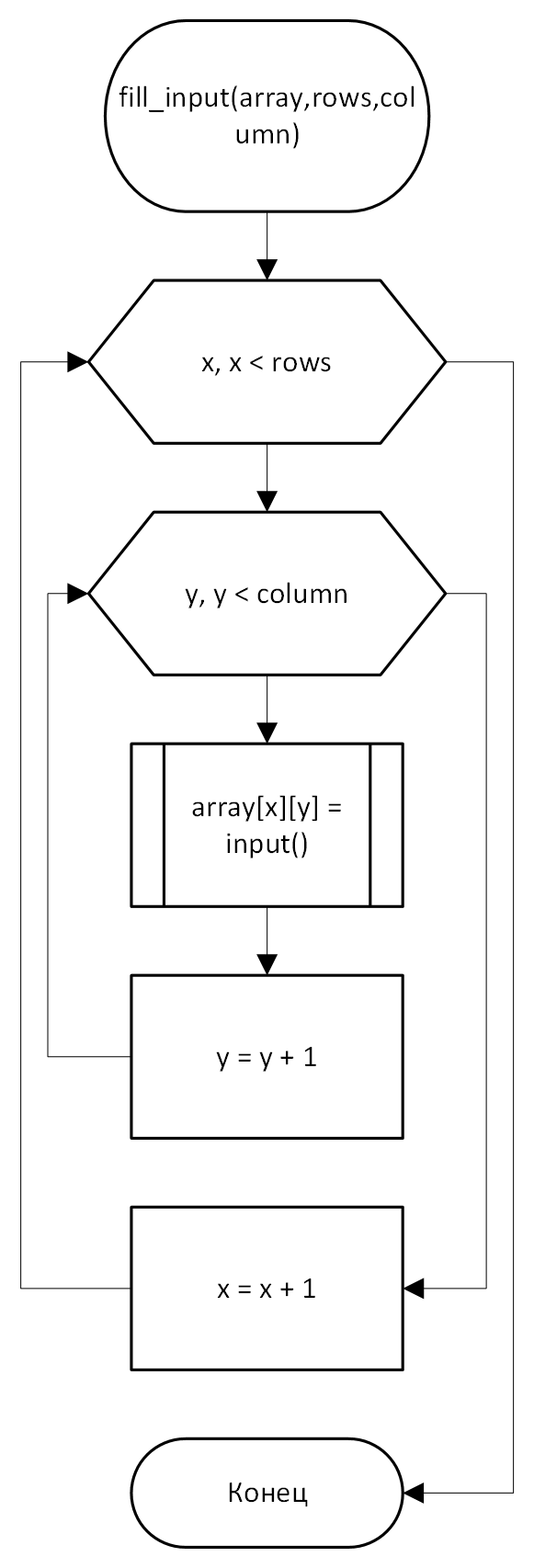
Рисунок 41 — Блок-схема функции input

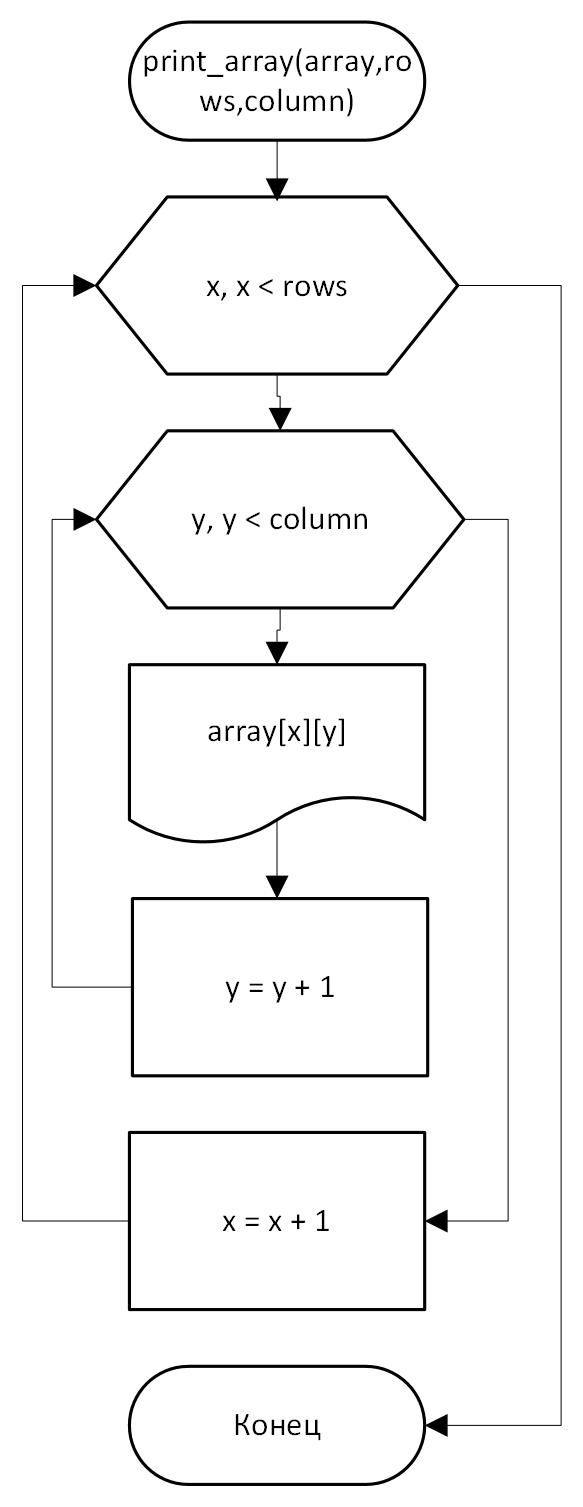
Рисунок 42 — Блок-схема функции create\_array

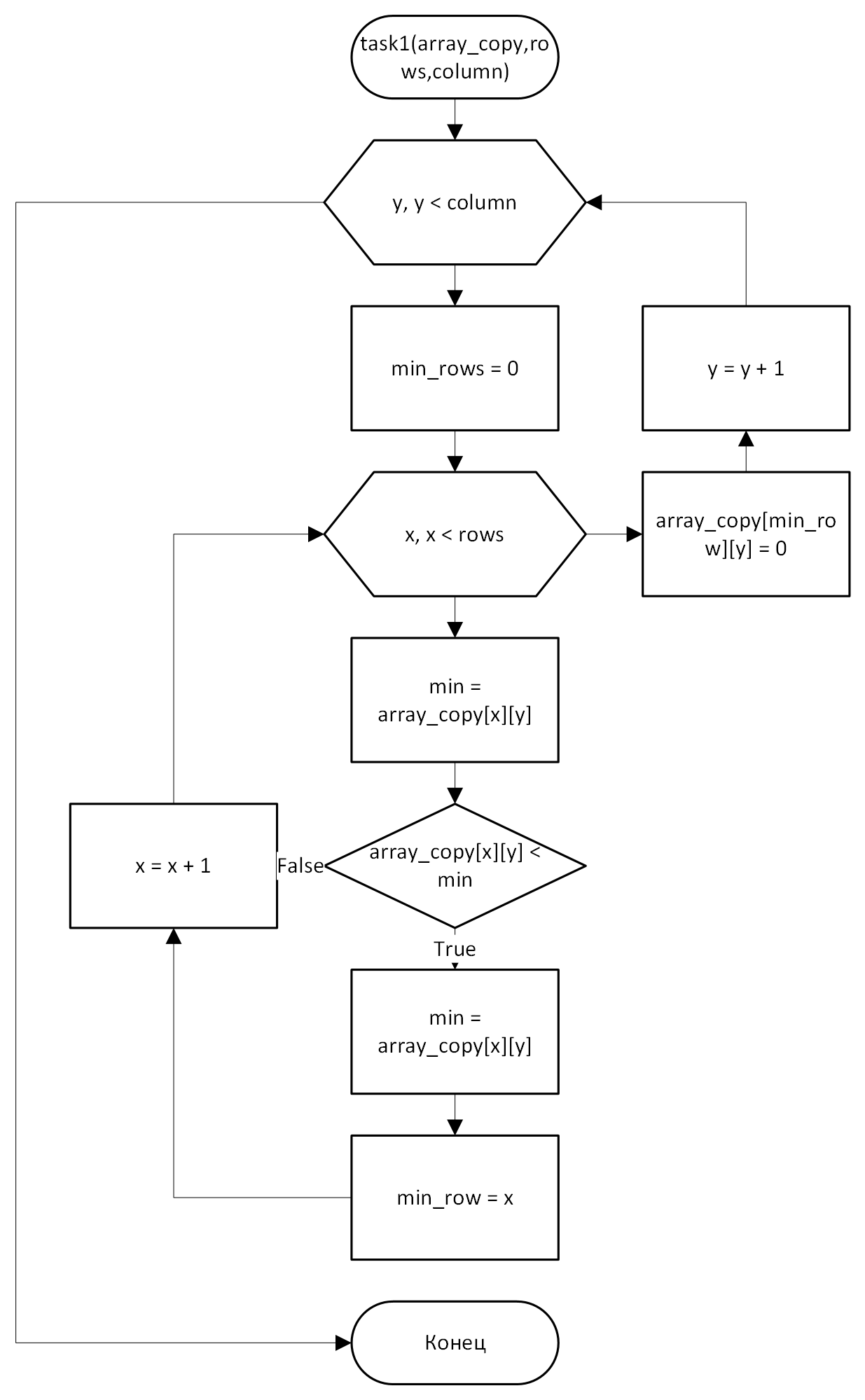
Рисунок 43 — Блок-схема функции copy\_array

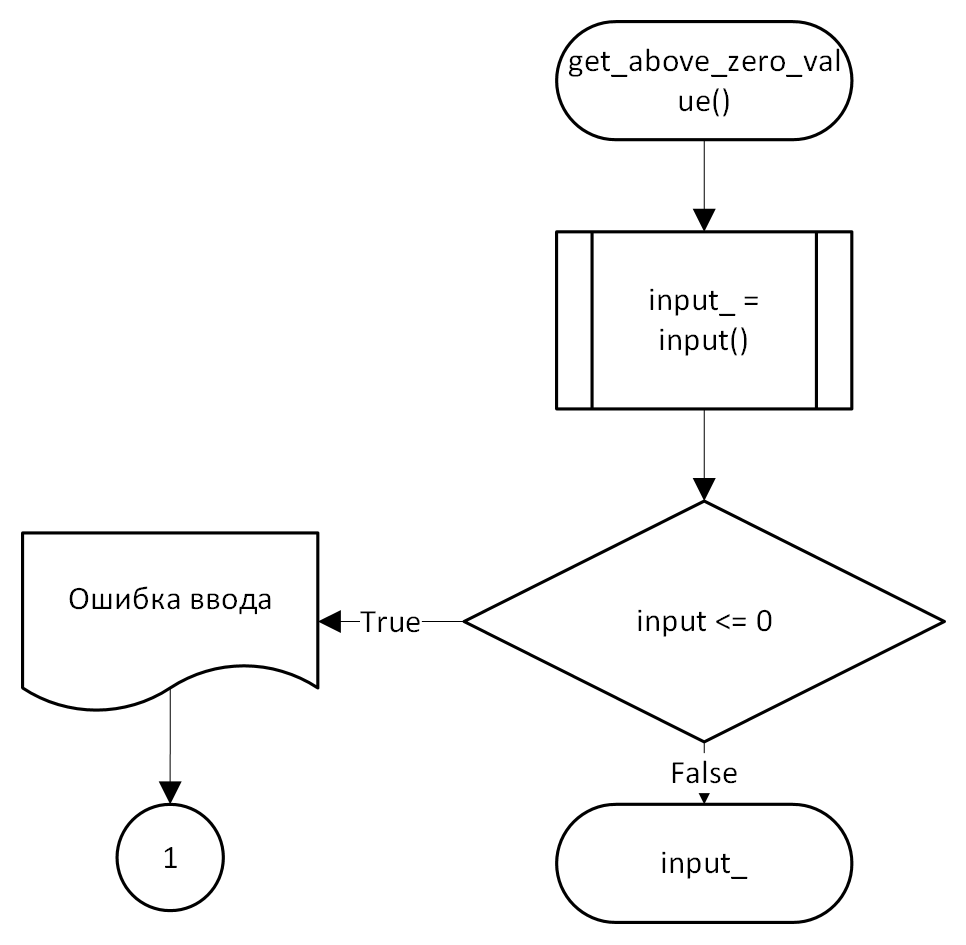
Рисунок 44 — Блок-схема функции check\_array

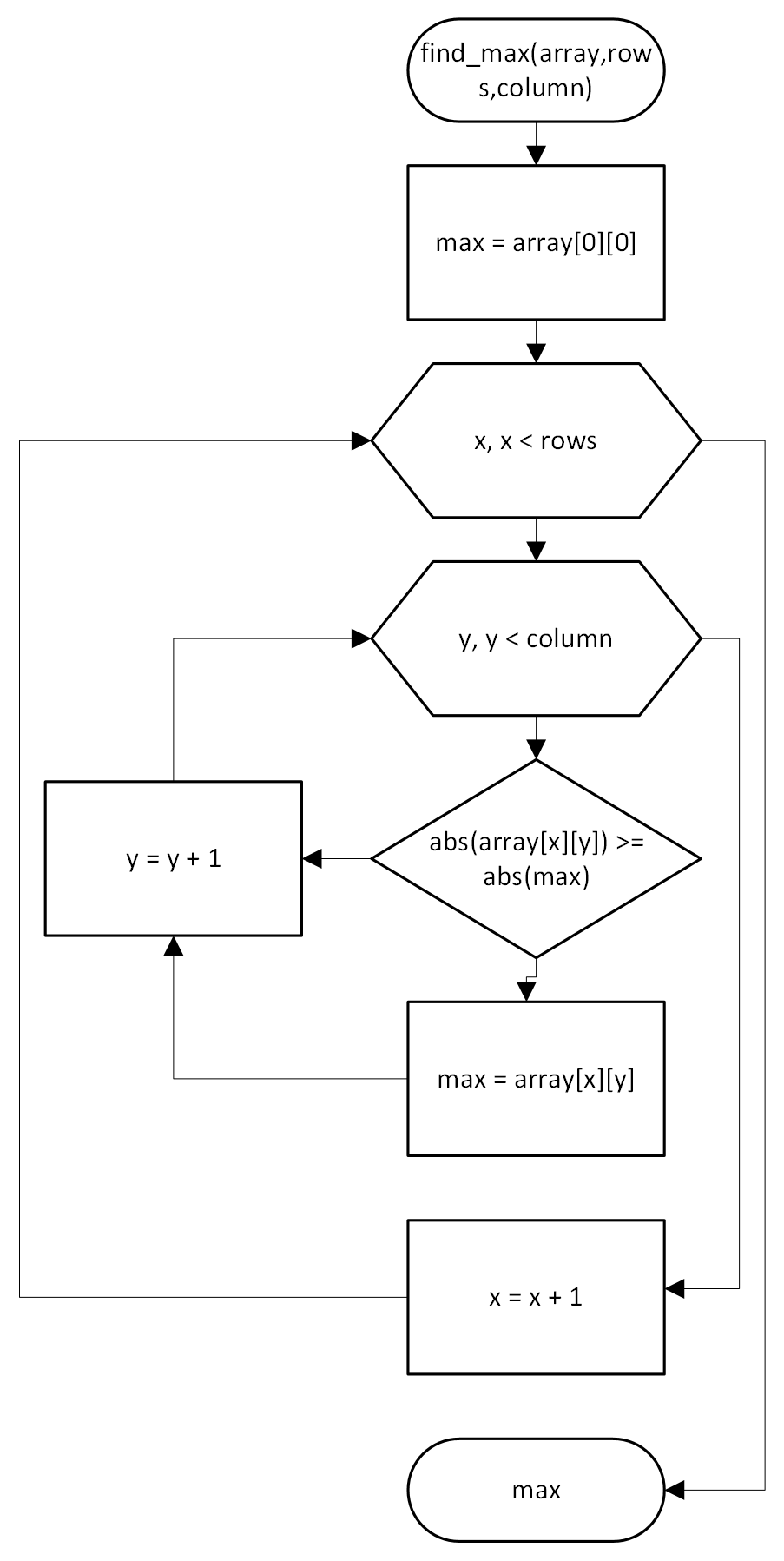
Рисунок 45 — Блок-схема функции fill\_random

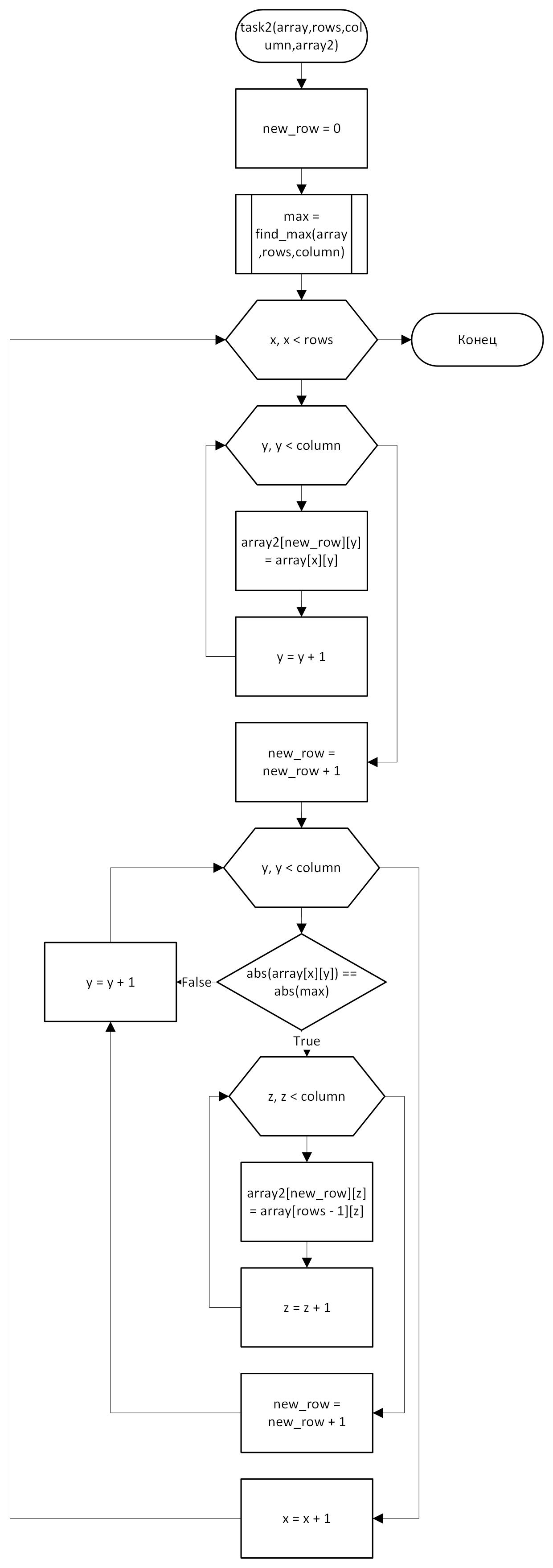
Рисунок 46 — Блок-схема функции fill\_input

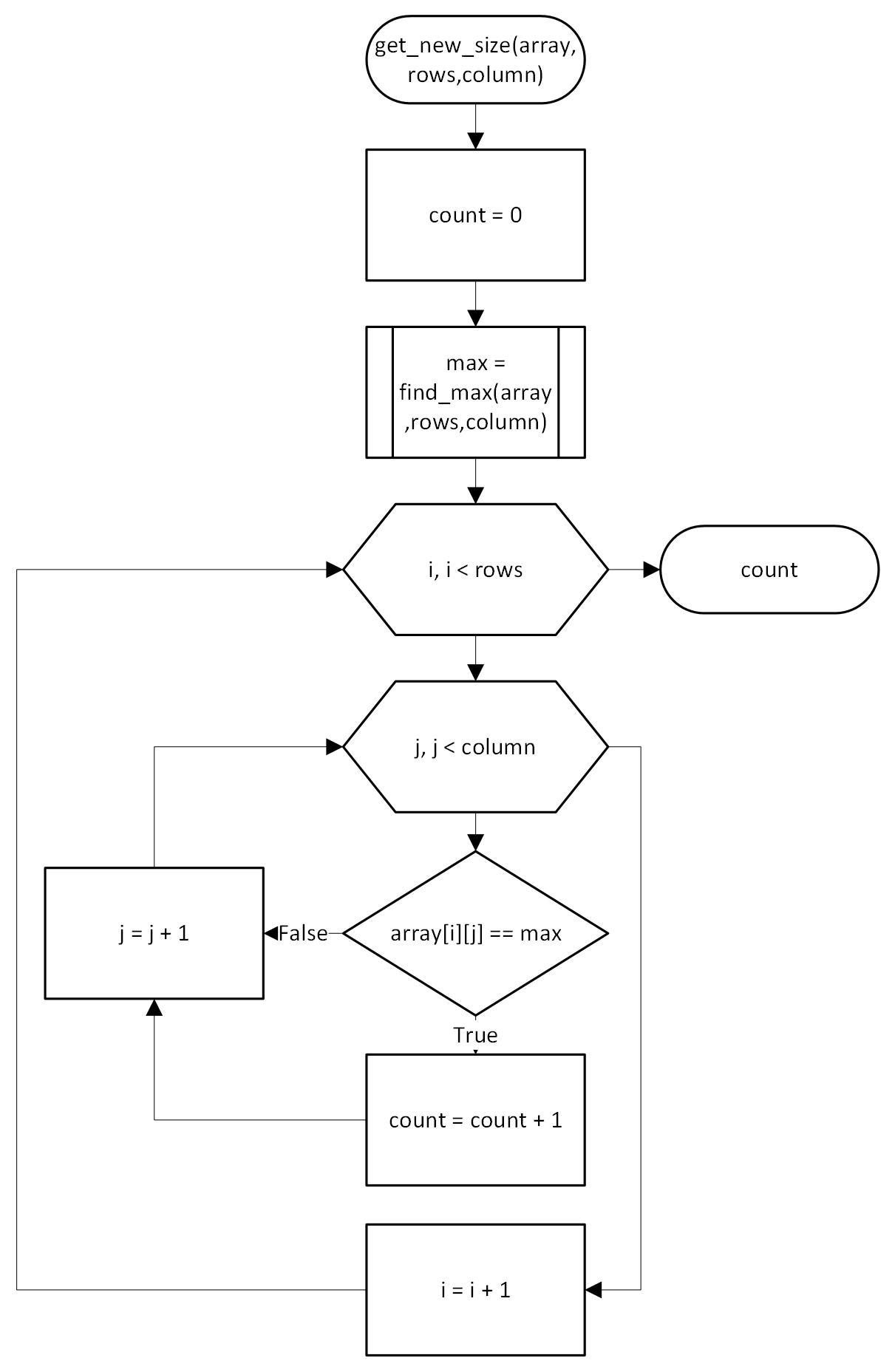
Рисунок 47 — Блок-схема функции print\_array

Рисунок 48 — Блок-схема функции task1

Рисунок 49 - Блок-схема функции get\_above\_zero\_value

Рисунок 50 — Блок-схема функции find\_max

Рисунок 51 — Блок-схема функции task2

Рисунок 52 — Блок-схема функции get\_new\_size

## **3.3 Текст программы на языке С**

#include <stdio.h>

#include <time.h>

#include <stdlib.h>

/\*\*

\* @brief Функция ввода целого числа.

\* @return целое число.

\*/

int input(void);

/\*\*

\* @brief Функция выделения памяти.

\* @param rows значение параметра rows.

\* @param column значение параметра column.

\* @return массив.

\*/

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

/\*\*

\* @brief Функция копирования массива.

\* @param array передаёт массив array.

\* @param rows значение параметра rows.

\* @param column значение параметра column.

\* @return массив.

\*/

int\*\* copy\_array(const int\*\* array, const size\_t rows, const size\_t column);

/\*\*

\* @brief Функция проверки массива на пустоту.

\* @param arr указатель на массив.

\*/

void check\_array(const int\*\* arr);

/\*\*

\* @brief Функция заполнения массива случайными числами.

\* @param array передаёт массив array.

\* @param rows значение параметра rows.

\* @param column значение параметра column.

\*/

void fill\_random(int\*\* array, const size\_t rows, const size\_t column,const int low\_value,const int high\_value);

/\*\*

\* @brief Функция заполнения массива вводимыми числами.

\* @param array передаёт массив array.

\* @param rows значение параметра rows.

\* @param column значение параметра column.

\*/

void fill\_input(int\*\* array, const size\_t rows, const size\_t column);

/\*\*

\* @brief Функция-нумератор.

\* @param input вызов функции ввода.

\* @param random вызов функции заполнения.

\*/

enum Task

{

user\_input = 1,

random\_input,

};

/\*\*

\* @brief Функция вывода массива.

\* @param array передаёт массив.

\* @param rows количество строк.

\* @param column количество столбцов.

\*/

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

/\*\*

\* @brief Функция заполнения массива

\* @param command - переменная команды заданной пользователем

\* @param array - наш массив

\* @param rows - строки массива

\* @param column - столбцы массива

\*/

void fill\_array(const int command, int\*\* array, const size\_t rows, const size\_t column,const int low\_value,const int high\_value);

/\*\*

\* @brief Функция выполнения задания 1

\* @param array\_copy - копия массива для безопасного взаимодейтсвия

\* @param rows - строки массива

\* @param column - столбцы массива

\* @remarksВыполняет заданеи 1

\*/

void task1(int\*\* array\_copy, const size\_t rows, const size\_t column);

/\*\*

\* @brief Функция нахождения максимального значения в массиве

\* @param array - наш массив

\* @param rows - строки массива

\* @param column - столбцы массива

\* return возвращает максимальное значение в массиве

\*/

int find\_max(const int\*\* array, const size\_t rows, const size\_t column);

/\*\*

\* @brief Функция выполнения задания 2

\* @param array - исходный массив

\* @param rows - строки исходного массива

\* @param column - столбцы исходного массива

\* @param array2 - новый расширенный массив

\* @param column2 - рассширенное кол-во столбцов

\* @param max - максимальное значение из массива

\* @param actual\_rows - фактическое количество строк

\* @remarks Выполняет задание 2

\*/

void task2(int\*\* array, const size\_t rows, const size\_t column, int\*\* array2);

/\*\*

\* @brief Функция полной чистки массива

\* @param array - наш массив

\* @param rows - строки массива

\*/

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

/\*\*

\* @brief Функция проверки введенного значени

\* @remarks если значение 0 и меньше возвращает код ошибки /c EXIT\_FAILURE

\*/

size\_t get\_above\_zero\_value(void);

/\*\*

\* @brief Функция счета нового количества строк

\* @return Возвращает количество строк, которые удовлетворяют по заданию 2

\*/

int get\_new\_size(const int\*\* array,const size\_t rows,const int column);

/\*\*

\* @brief Точка входа в программу.

\* @return Возвращает в случае успеха.

\*/

int main(void)

{

puts("Rows:");

size\_t rows = get\_above\_zero\_value();

puts("Column:");

size\_t column = get\_above\_zero\_value();

puts("Enter low and high value of array");

int low\_value = input(), high\_value = input();

printf("%d for user input, %d for random input", user\_input, random\_input);

int command = input();

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

fill\_array(command, array, rows, column,low\_value,high\_value);

print\_array(array, rows, column);

puts("Task1");

int\*\* array\_copy1 = copy\_array(array, rows, column);

task1(array\_copy1, rows, column);

print\_array(array\_copy1, rows, column);

free\_array(array\_copy1,rows);

puts("Task2");

size\_t actual\_rows = rows + get\_new\_size(array,rows,column);

int\*\* array2 = create\_array(actual\_rows, column);

task2(array, rows, column, array2);

print\_array(array2, actual\_rows, column);

print\_array(array2, actual\_rows, column);

free\_array(array, rows);

free\_array(array2, actual\_rows);

return 0;

}

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

{

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

{

free(array[i]);

}

free(array);

}

void fill\_array(const int command, int\*\* array, const size\_t rows, const size\_t column,const int low\_value,const int high\_value)

{

switch ((enum Task)command)

{

case user\_input: fill\_input(array, rows, column); break;

case random\_input: fill\_random(array, rows, column,low\_value,high\_value); break;

default: printf("Incorrect value"); exit(EXIT\_FAILURE);

}

}

int input()

{

int input = 0;

if (scanf("%d", &input) != 1)

{

printf("Incorrect value");

exit(EXIT\_FAILURE);

}

return input;

}

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

{

int\*\* array = (int\*\*)malloc(sizeof(int\*) \* rows);

check\_array(array);

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

{

array[i] = (int\*)malloc(sizeof(int) \* column);

check\_array(array[i]);

}

return array;

}

int\*\* copy\_array(const int\*\* array, const size\_t rows, const size\_t column)

{

check\_array(array);

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

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

{

for (size\_t y = 0; y < column; y++)

{

array\_copy[x][y] = array[x][y];

}

}

return array\_copy;

}

void check\_array(const int\*\* arr)

{

if (arr == NULL)

{

printf("Memory not allocated.\n");

exit(EXIT\_FAILURE);

}

}

void fill\_random(int\*\* array, const size\_t rows, const size\_t column,const int low\_value, const int high\_value)

{

srand(time(NULL));

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

{

for (size\_t y = 0; y < column; y++)

{

array[x][y] = rand() % (high\_value - low\_value + 1) + low\_value;

}

}

}

void fill\_input(int\*\* array, const size\_t rows, const size\_t column)

{

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

{

for (size\_t y = 0; y < column; y++)

{

array[x][y] = input();

}

}

}

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

{

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

{

for (size\_t y = 0; y < column; y++)

{

printf("%d ", array[x][y]);

}

printf("\n");

}

}

void task1(int\*\* array\_copy, const size\_t rows, const size\_t columns)

{

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

{

size\_t min\_row = 0;

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

{

int min = array\_copy[x][y];

if (array\_copy[x][y] < min)

{

min = array\_copy[x][y];

min\_row = x;

}

}

array\_copy[min\_row][y] = 0;

}

}

size\_t get\_above\_zero\_value(void)

{

int input\_ = input();

if (input\_ <= 0)

{

puts("Input error");

exit(EXIT\_FAILURE);

}

return (size\_t)input\_;

}

int find\_max(const int\*\* array, const size\_t rows, const size\_t column)

{

int max = array[0][0];

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

{

for (size\_t y = 0; y < column; y++)

{

if (abs(array[x][y]) >= abs(max))

{

max = array[x][y];

}

}

}

return max;

}

void task2(int\*\* array, const size\_t rows, const size\_t column, int\*\* array2)

{

size\_t new\_row = 0;

int max = find\_max (array, rows, column);

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

{

for (size\_t y = 0; y < column; y++)

{

array2[new\_row][y] = array[x][y];

}

new\_row++;

for (size\_t y = 0; y < column; y++)

{

if (abs(array[x][y]) == abs(max))

{

for (size\_t z = 0; z < column; z++)

{

array2[new\_row][z] = array[rows - 1][z];

}

new\_row++;

break;

}

}

}

}

int get\_new\_size(const int\*\* array,const size\_t rows,const int column)

{

int count = 0;

int max = find\_max(array, rows, column);

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

{

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

{

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

{

count++;

break;

}

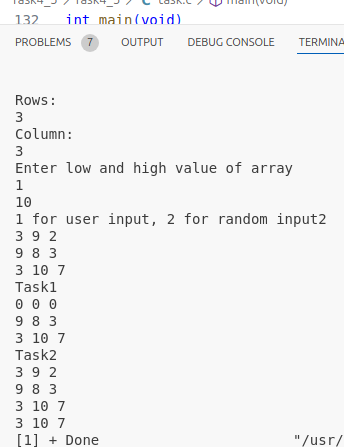
}

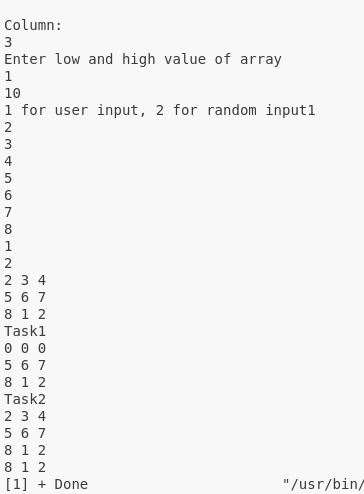
}

return count;

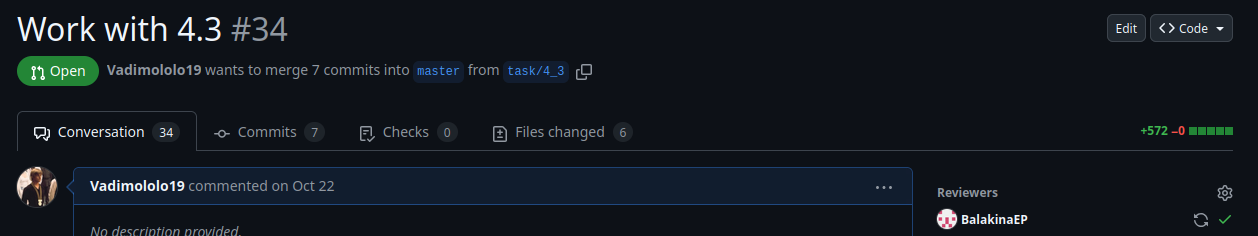
}

## **3.4 Результаты выполнения программы**

Рисунок 53 — Результат выполнения программы со случайными числами

Рисунок 54 — Результат выполнения программы с числами пользователя

## **3.5 Отметка о выполнении задания в системе контроля версий**

Рисунок 55 — Отметка о выполнении