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 2

Выполнил: ст. гр. ТКИ - 142

Бахров Кирилл Сергеевич

Проверил: к.т.н., доц. Васильева М. А Проверил: к.т.н., доц. Балакина Е. П.

Москва 2023

**Оглавление**

**Решение задачи 4.14**

Формулировка задания4

Блок-схема алгоритма5-7

Текст программы на языке C8-10

Результат выполнения программы11

Выполнение тестовых примеров12

Отметка о выполнении задания в веб-хостинге системы контроля версий13

**Решение задачи 4.214**

Формулировка задания14

Блок-схема алгоритма15

Текст программы на языке C16

Результат выполнения программы17

Выполнение тестовых примеров18

Отметка о выполнении задания в веб-хостинге системы контроля версий19

**Решение задачи 4.3**20

Формулировка задания20

Блок-схема алгоритма21-22

Текст программы на языке C23-26

Результат выполнения программы27

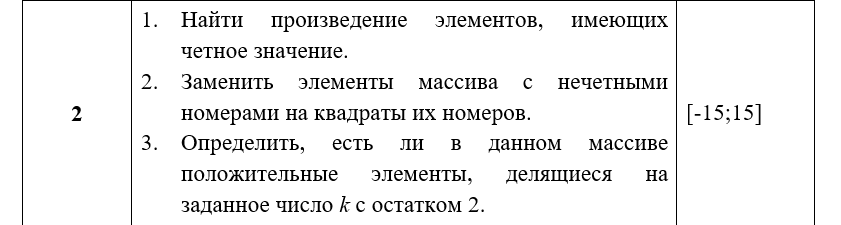
Выполнение тестовых примеров28-29

Отметка о выполнении задания в веб-хостинге системы контроля версий30

1. РЕШЕНИЕ ЗАДАЧИ 4.1
   1. ФОРМУЛИРОВКА ЗАДАНИЯ

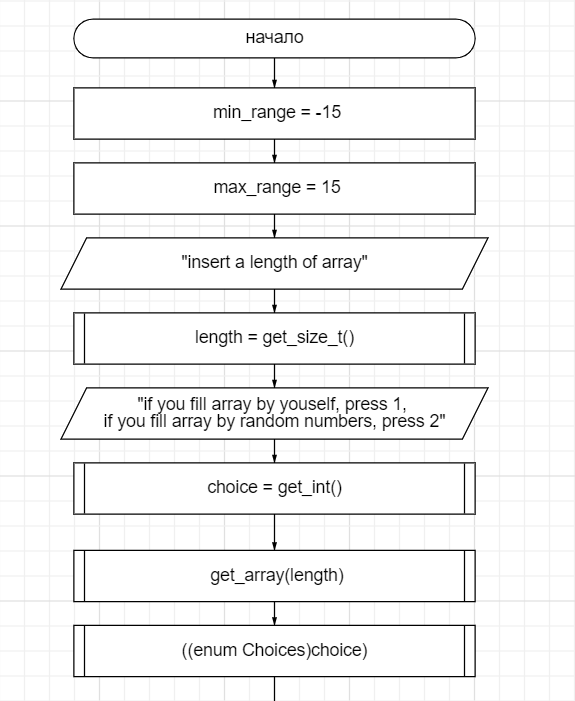
Создать одномерный массив из *n* целых чисел. Реализовать возможность заполнения массива, как случайными числами, так и с помощью клавиатуры по желанию пользователя. Пункты задания организовать в виде функций (методов). Составить блок-схему.

Таблица  – Исходные данные



* 1. БЛОК-СХЕМА АЛГОРИТМА

Блок-схема основного алгоритма представлена ниже (Рисунок 1). Блок-схемы используемых функций представлены ниже (Рисунки 2-5).

  
Рисунок 1 ­ Блок-схема основного алгоритма

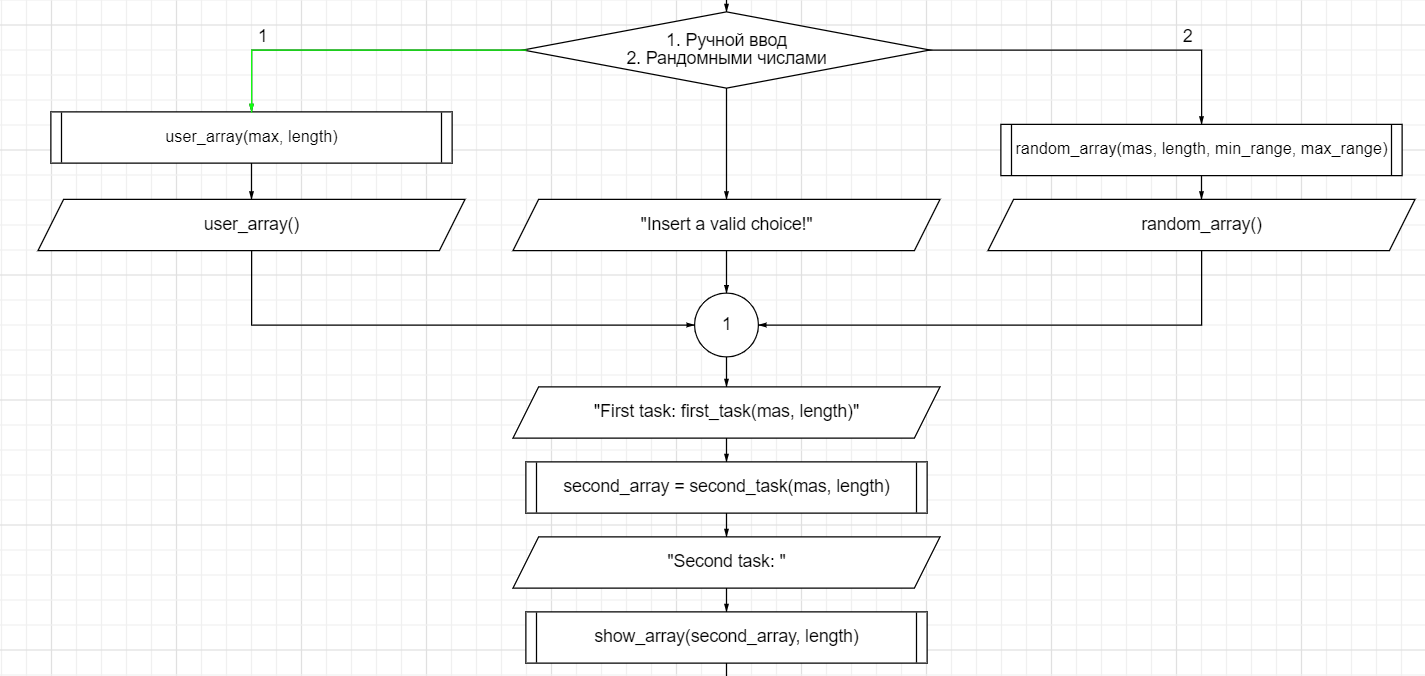


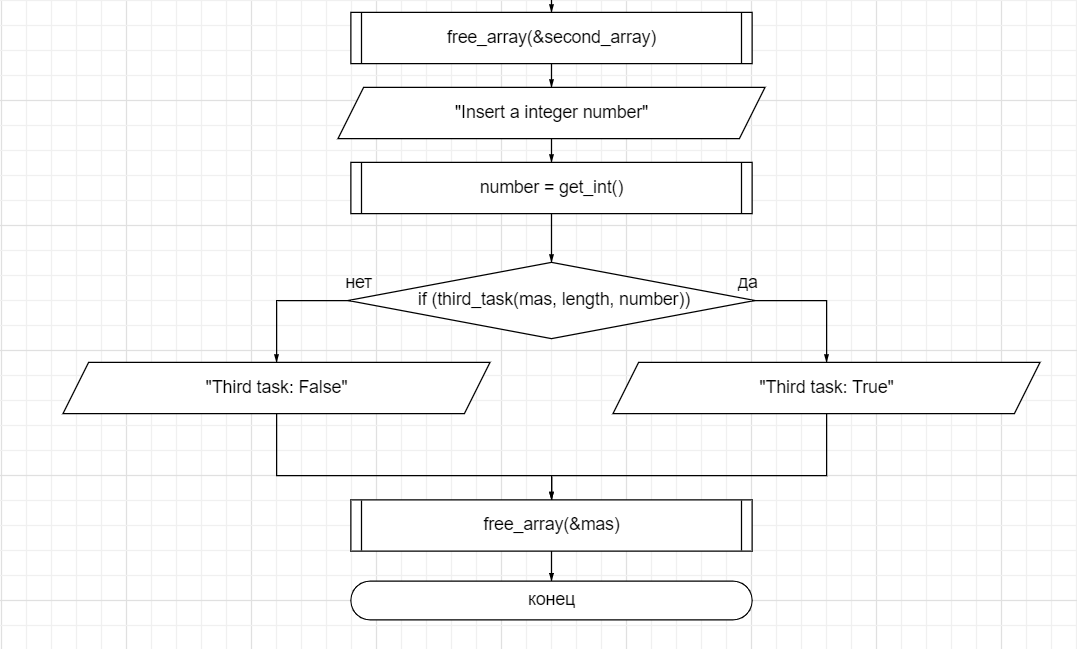
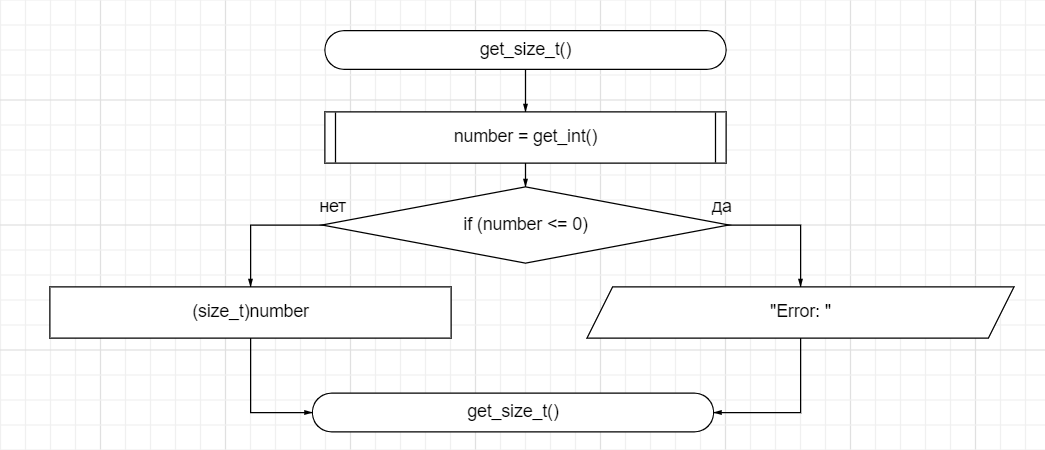
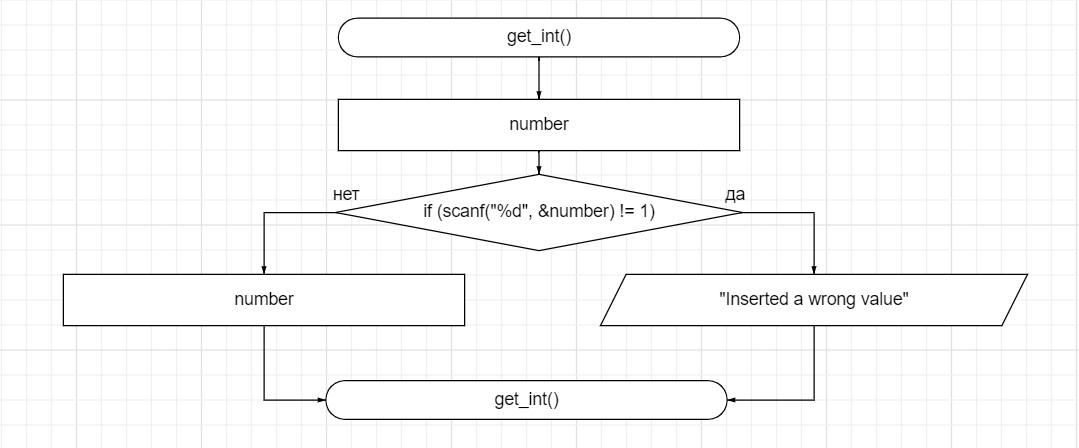
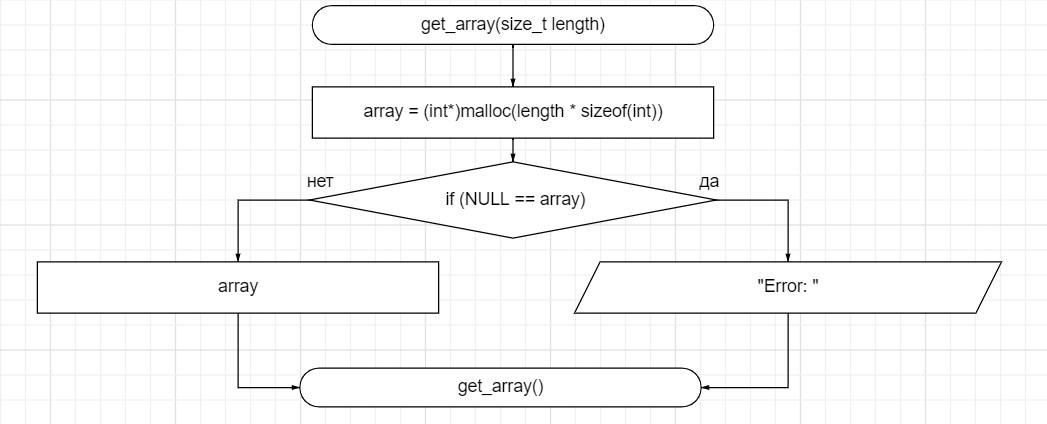
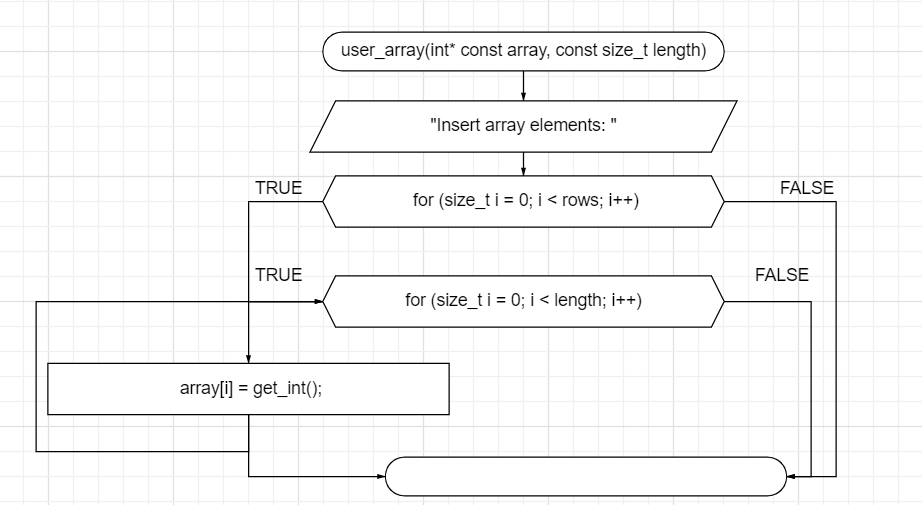
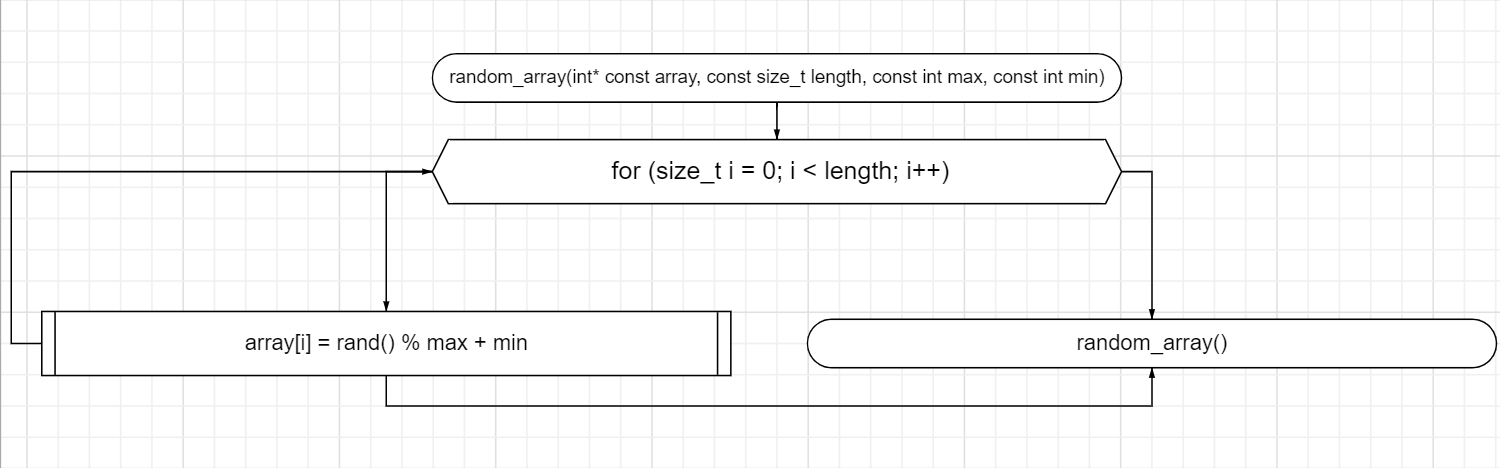
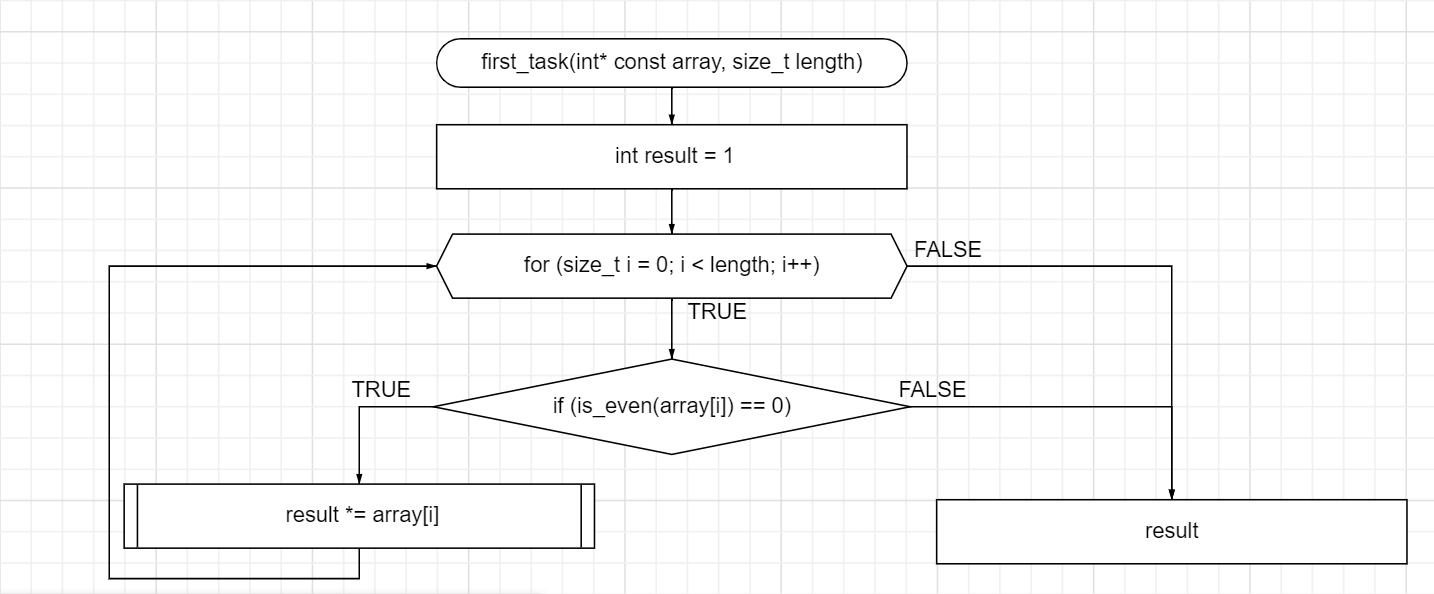
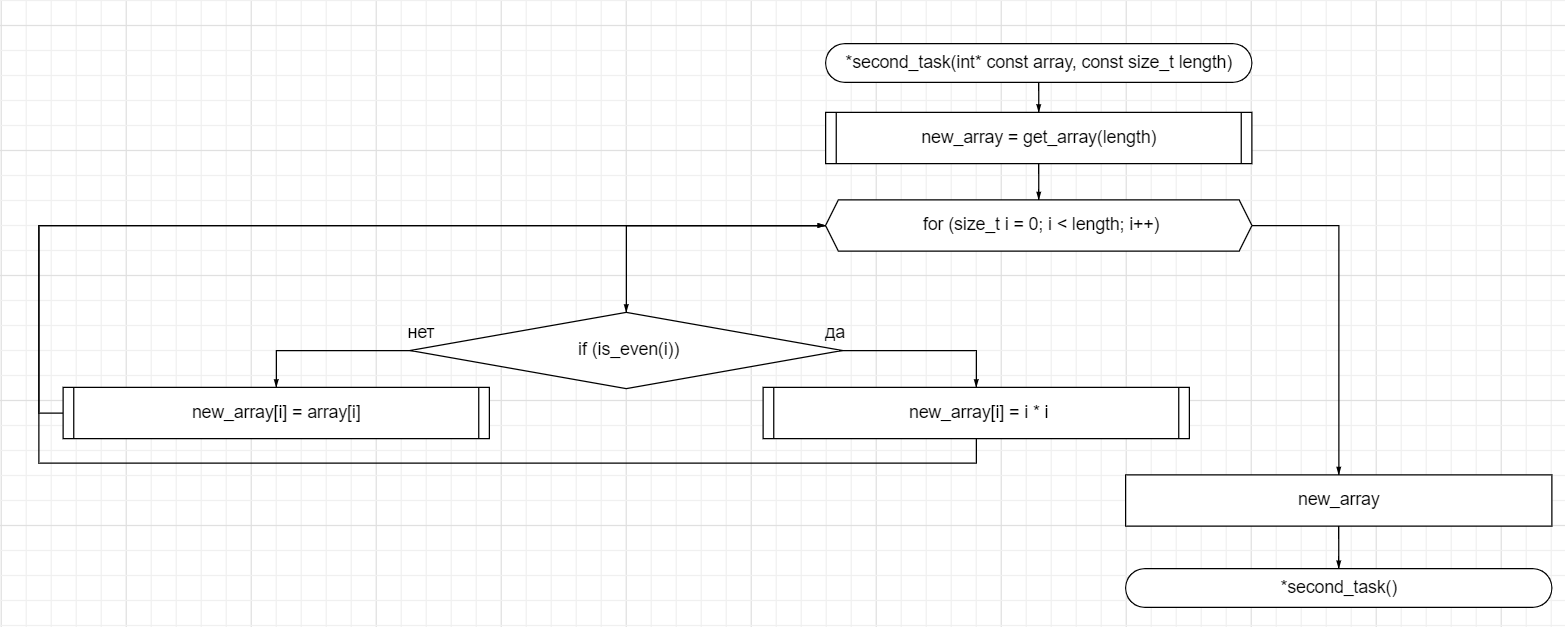
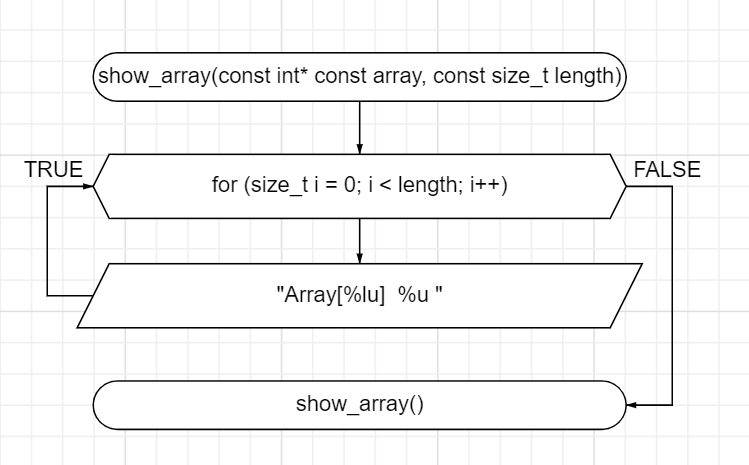
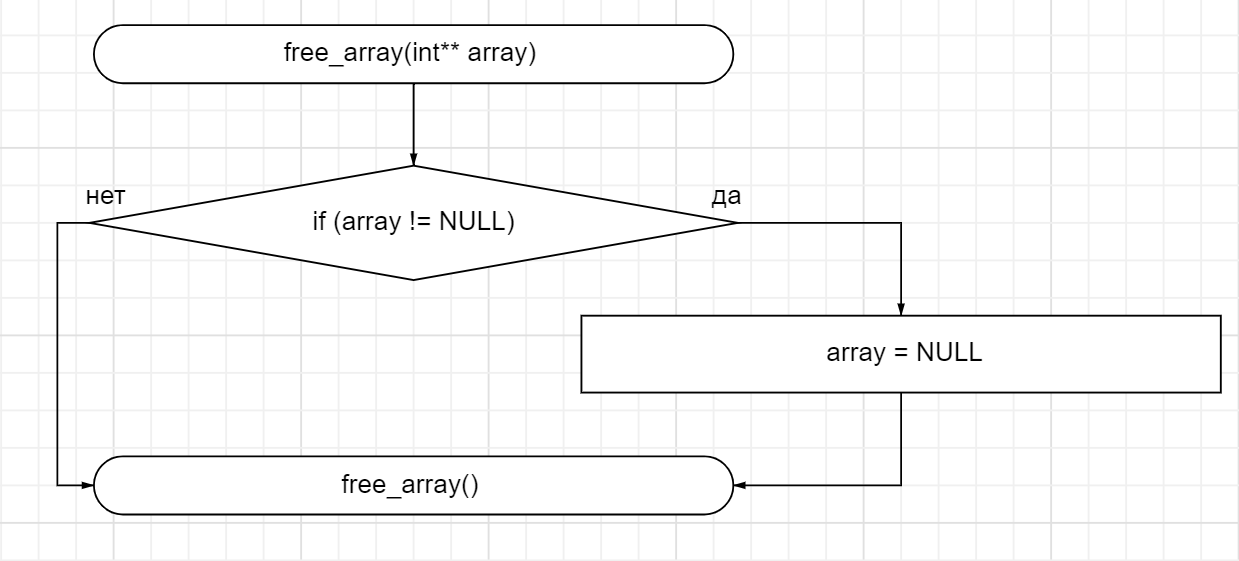
Рисунок 2 ­ Блок-схема основного алгоритма  
  
  
  
Рисунок 3 ­ Блок-схема основного алгоритма  
  
 Рисунок 4 – Блок-схема используемых функций  
  
Рисунок 5 – Блок-схема используемых функций  
   
Рисунок 6 – Блок-схема используемых функций   
   
Рисунок 7 – Блок-схема используемых   


Рисунок 8 Блок-схема используемых

  
 Рисунок 9 – Блок-схема используемых   
   
Рисунок 10 – Блок-схема используемых   
   
Рисунок 11 – Блок-схема используемых

   
Рисунок 12 – Блок-схема используемых

* 1. ТЕКСТ ПРОГРАММЫ НА ЯЗЫКЕ C

#include <stdio.h>

#include <stdlib.h>

#include <malloc.h>

#include <errno.h>

#include <stdbool.h>

/\*\*

\* @brief Функция присваивает целочисленное значение переменной

\* @return целочисленное число

\*/

int get\_int();

/\*\*

\* @brief Функция присваивает переменной целочисленное значение и проверяет его на положителность

\* @return number положительное число

\*/

size\_t get\_size\_t();

/\*\*

\* @brief структура хранит константы, указывающие выбор пользователя заполнить массив

\* @brief user\_choice хранит значение, вызывающее ввод массива вручную

\* @brief random\_choice хранит значение, вызывающее ввод массива случайными числами

\*/

enum Choices

{

user\_choice = 1,

random\_choice = 2,

};

/\*\*

\* @brief Функция выделяет память под массив

\* @param length длина массива

\* @return указатель на пустой массив

\*/

int\* get\_array(size\_t length);

/\*\*

\* @brief Функция, показывающая массив поэлементно

\* @param array ссылка на показываемый массив

\* @param length длина массива

\*/

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

/\*\*

\* @brief Функция проверяет на четность целое число

\* @param number число

\* @return 0 четный

\* @return 1 нечетный

\*/

bool is\_even(int number);

/\*\*

\* @brief Функция заполняет массив, элементы вводит пользователь

\* @param array указатель на заполняемый массив

\* @param length длина массива

\*/

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

/\*\*

\* @brief Функция заполняет массив рандомными числами в диапазоне [min:max]

\* @param array указатель на заполняемый массив

\* @param min минимальное значение диапазона

\* @param max максимальное значение

\* @param length длина массива

\*/

void random\_array(int\* array, const size\_t length, const int max, const int min);

/\*\*

\* @brief Находит произведение элементов, имеющих четное значение

\* @param array указатель на массив

\* @param length размерность массива

\* @return результат

\*/

int first\_task(int\* const array, size\_t length);

/\*\*

\* @brief Заменяет элементы массива с нечетными номерами на квадраты их номеров.

\* @param array указатель на массив

\* @param length размерность массива

\* @return new\_array новый массив,удовлетворяющим данным требованиям

\*/

int \*second\_task(int\* const array, const size\_t length);

/\*\*

\* @brief Определяет, есть ли в данном массиве положительные элементы, делящиеся на заданное число k с остатком 2.

\* @param array указатель на массив

\* @param length размерность массива

\* @param number заданное число

\* @return true есть такие элементы

\* @return false нет таких элементов

\*/

bool third\_task(int\* const array, const size\_t length, const int number);

/\*\*

\* @brief Функция, освобождающая массив

\* @param array указатель на массив

\*/

void free\_array(int\*\* array);

/\*\*

\* @brief Точка входа в программу

\* @return 0 Программа исправна

\* @return 1 Программа досрочно закрылась с ошибкой

\*/

int main()

{

const int min\_range = -15;

const int max\_range = 15;

puts("insert a length of array\n");

size\_t length = get\_size\_t(); // size t

printf("if you fill array by youself, press %d, if you fill array by random numbers, press %d\n", (enum Choices)(user\_choice), (enum Choices)(random\_choice));

int choice = get\_int();

int\* mas = get\_array(length);

switch ((enum Choices)choice)

{

case random\_choice:

random\_array(mas, length, min\_range, max\_range);

break;

case user\_choice:

user\_array(mas, length);

break;

default:

puts("Insert a valid choice!\n");

return 1;

}

printf("First task: %d\n", first\_task(mas, length));

int\* second\_array = second\_task(mas, length);

puts("Second task:\n");

show\_array(second\_array, length);

free\_array(&second\_array);

puts("insert a integer number\n");

int number = get\_int();

if (third\_task(mas, length, number))

{

puts("Third task: True\n");

}

else

{

puts("Third task: False\n");

}

free\_array(&mas);

return 0;

}

int get\_int()

{

int number;

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

{

puts("Inserted a wrong value\n");

abort();

}

return number;

}

size\_t get\_size\_t()

{

int number = get\_int();

if (number <= 0)

{

errno = ERANGE;

perror("Error: \n");

abort();

}

return (size\_t)number;

}

int\* get\_array(size\_t length)

{

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

if (NULL == array)

{

errno = ENOMEM;

perror("Error: \n");

abort();

}

return array;

}

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

{

puts("Insert array elements:\n");

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

{

array[i] = get\_int();

}

}

void random\_array(int\* const array, const size\_t length, const int max, const int min)

{

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

{

array[i] = rand() % max + min;

}

}

int first\_task(int\* const array, size\_t length)

{

int result = 1;

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

{

if (is\_even(array[i]) == 0)

{

result \*= array[i];

}

}

return result;

}

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

{

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

{

printf("Array[%lu] %u\n", i, array[i]);

}

}

int \*second\_task(int\* const array, const size\_t length)

{

int\* new\_array = get\_array(length);

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

{

if (is\_even(i))

{

new\_array[i] = i \* i;

}

else

{

new\_array[i] = array[i];

}

}

return new\_array;

}

bool third\_task(int\* const array, const size\_t length, const int number)

{

for (int i = 0; i < length; i++)

{

if (array[i] % number == 2)

{

return true;

}

}

return false;

}

bool is\_even(int number)

{

return number % 2 == 0;

}

void free\_array(int\*\* array)

{

if (array != NULL)

{

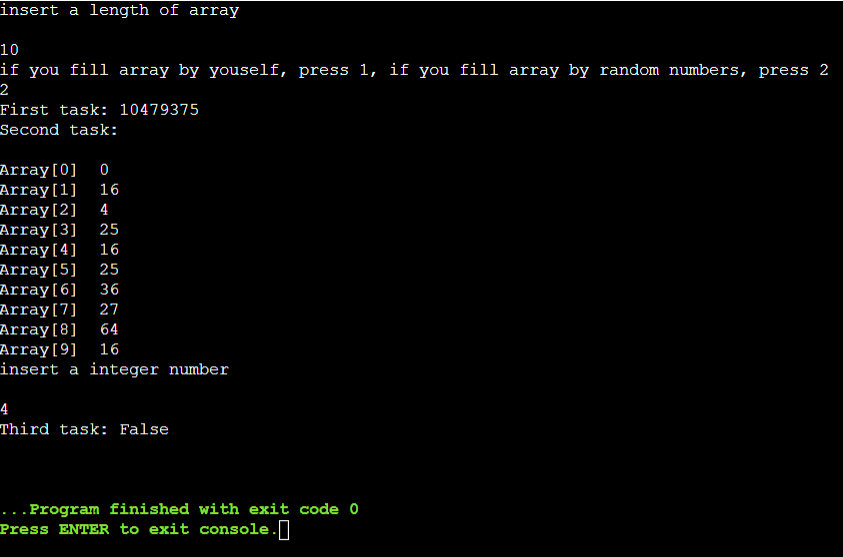
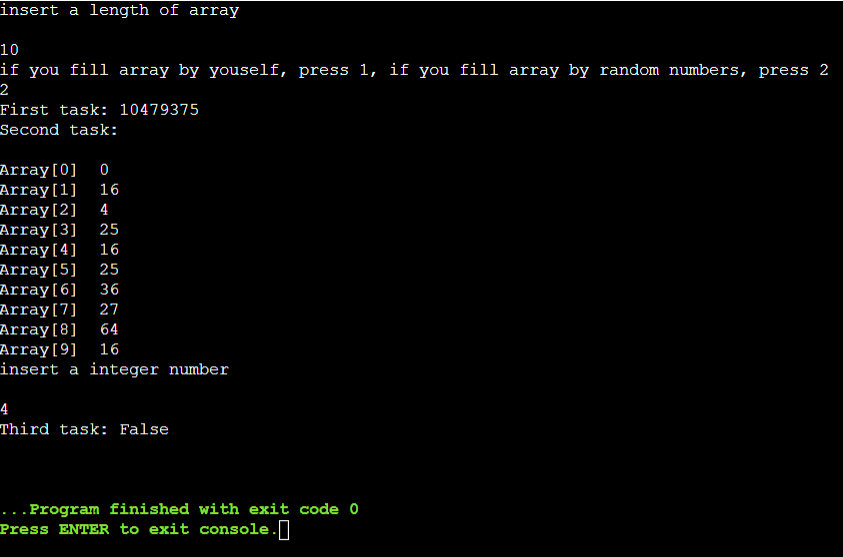
free(\*array);

array = NULL;

}

}

* 1. РЕЗУЛЬТАТЫ ВЫПОЛНЕНИЯ ПРОГРАММЫ

Результаты выполнения программы представлены ниже (Рисунок 13-14).  
  
Рисунок 13 – Результаты выполнения программы   
Рисунок 14 – Результаты выполнения программы

* 1. ВЫПОЛНЕНИЕ ТЕСТОВЫХ ПРИМЕРОВ
  2. ОТМЕТКА О ВЫПОЛНЕНИИ ЗАДАНИЯ В ВЕБ-ХОСТИНГЕ СИСТЕМЫ КОНТРОЛЯ ВЕРСИЙ

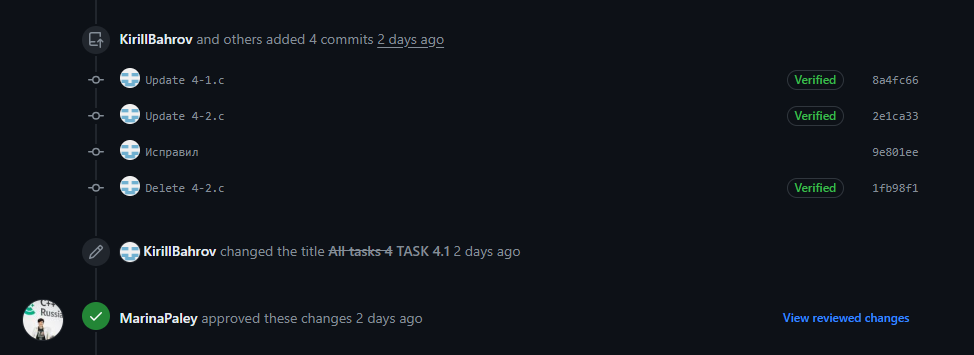
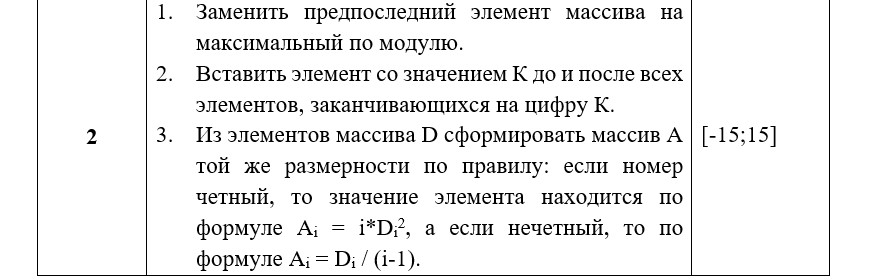


Рисунок – 15 Отметка о выполнении задания  
 в веб-хостинге системы контроля версий

1. РЕШЕНИЕ ЗАДАЧИ 4.2

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.  
  
Таблица 2 – Исходные данные



2.2 БЛОК-СХЕМА АЛГОРИТМА

Блок-схема основного алгоритма представлена ниже (Рисунок 16 - 21). Блок-схемы используемых функций представлены ниже (Рисунки 2-5).

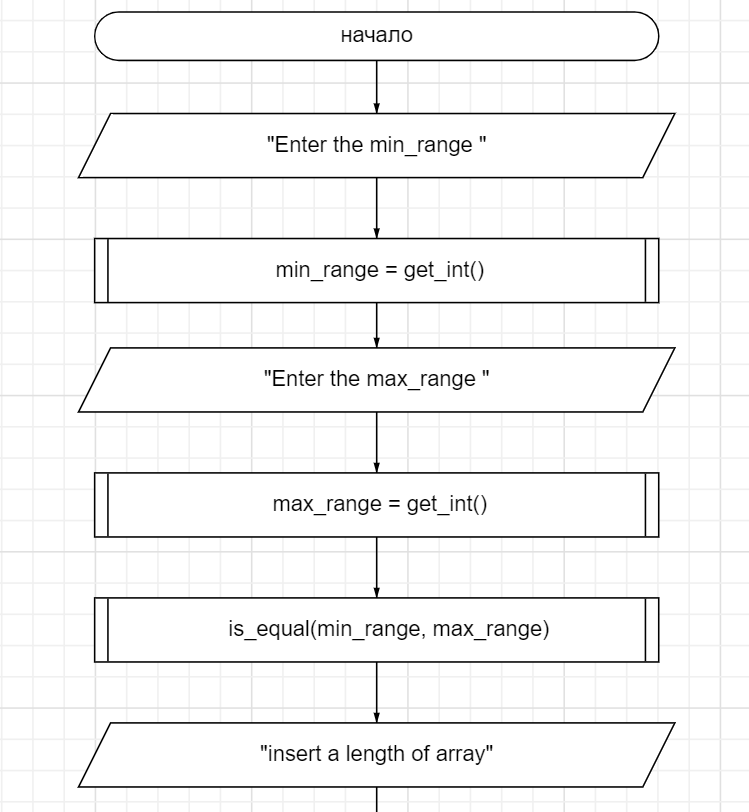


Рисунок 16 – Блок-схема основного алгоритма

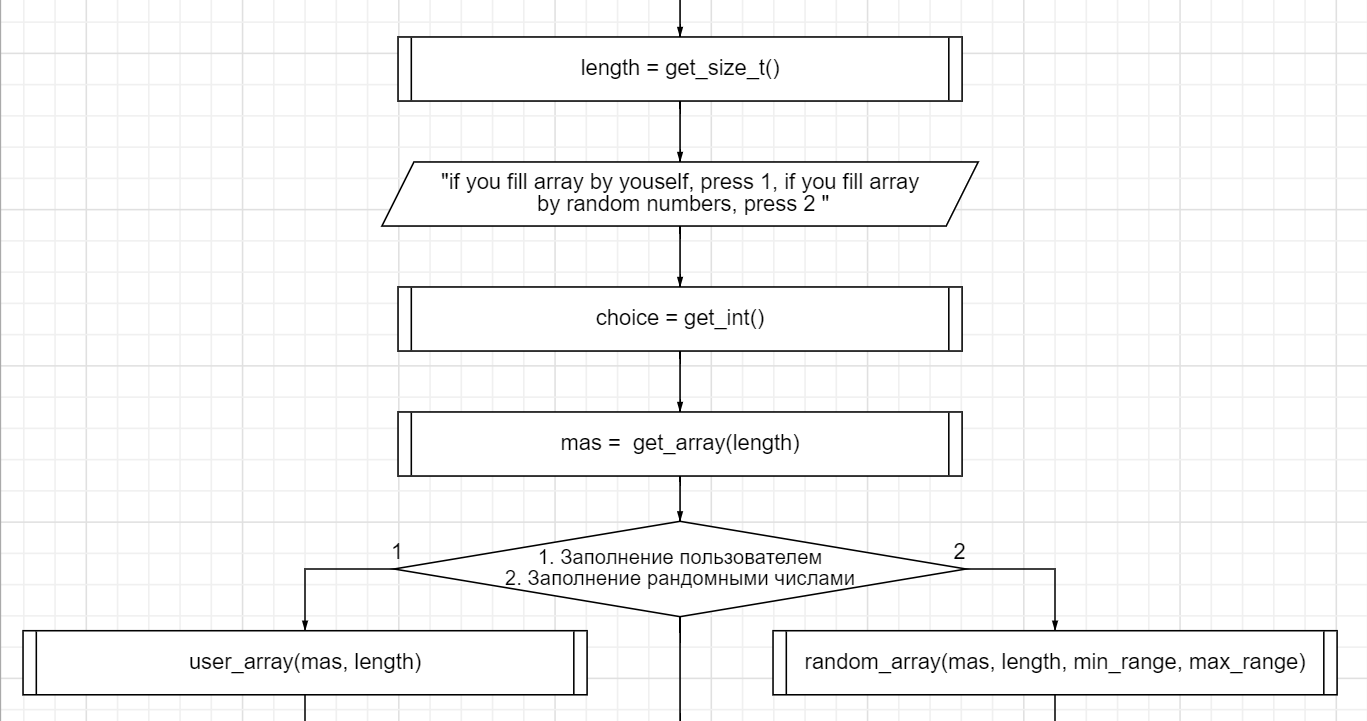
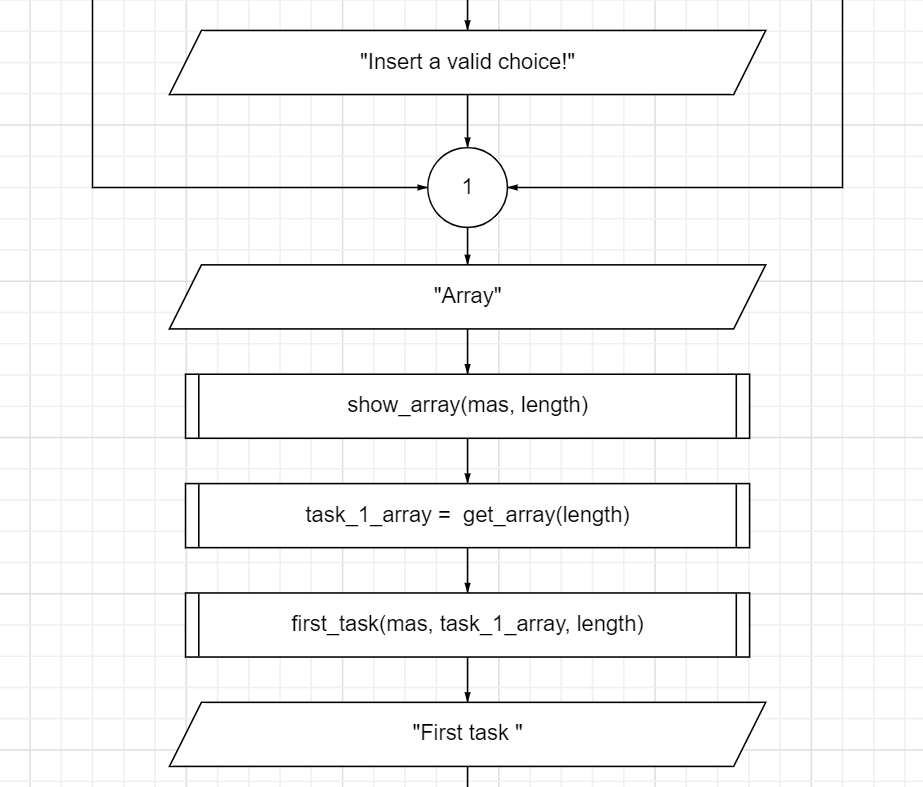


Рисунок 17 – Блок-схема основного алгоритма  
  
Рисунок 18 – Блок-схема основного алгоритма

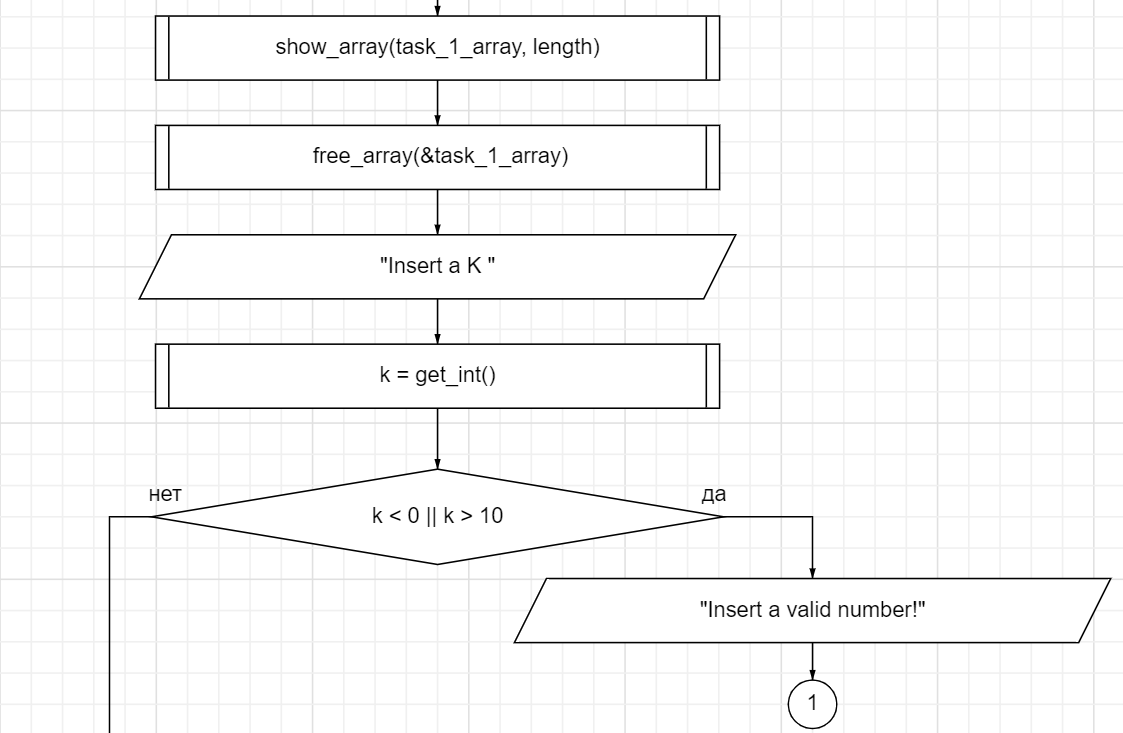


Рисунок 19 – Блок-схема основного алгоритма

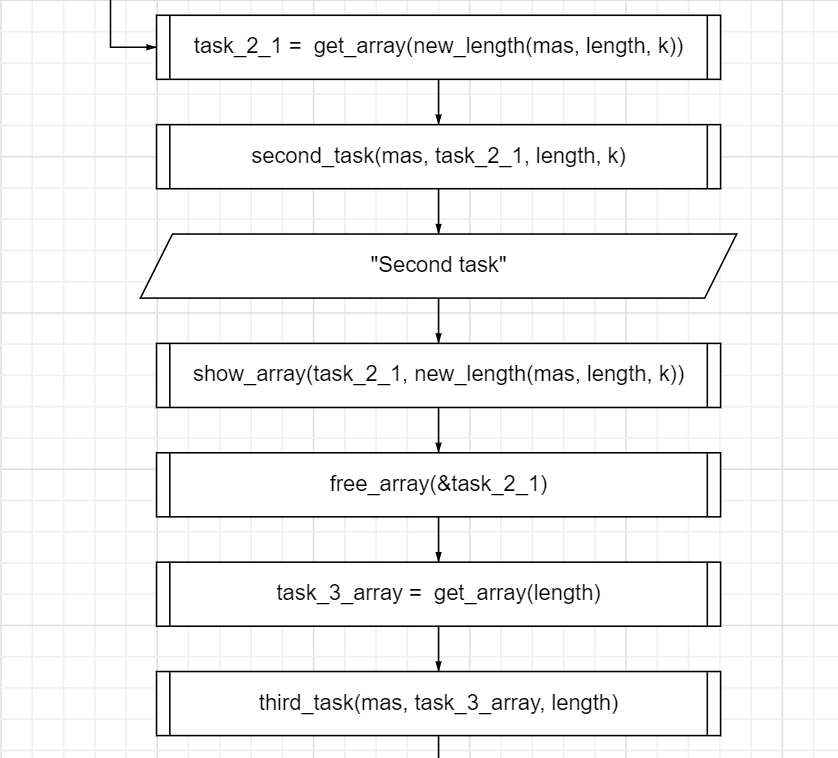


Рисунок 20 – Блок-схема основного алгоритма

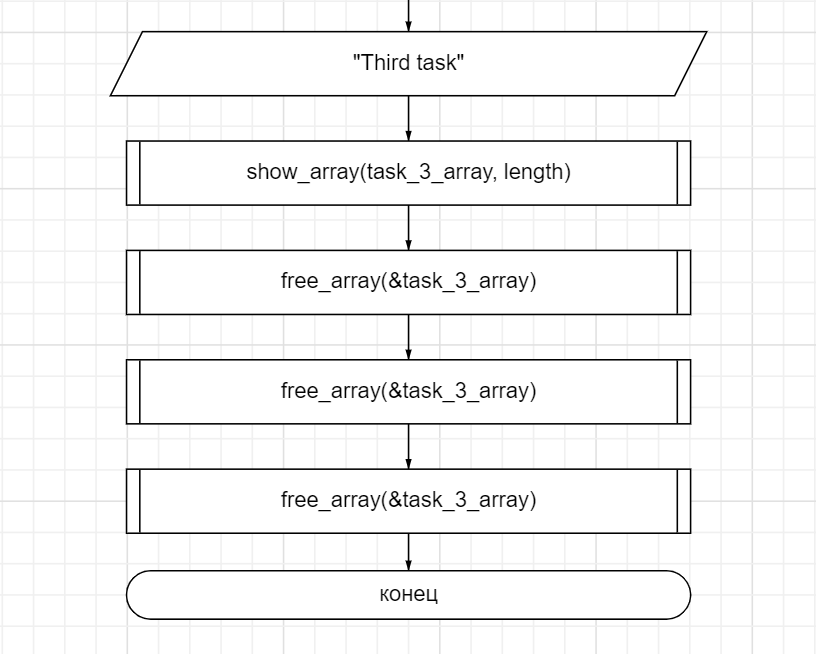
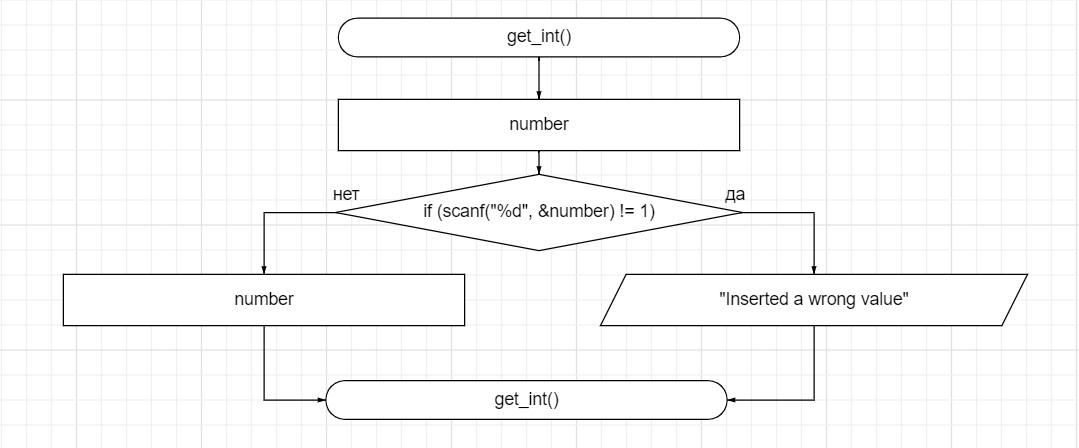


Рисунок 21 – Блок-схема основного алгоритма   
Рисунок 22 – Блок-схема используемых функций

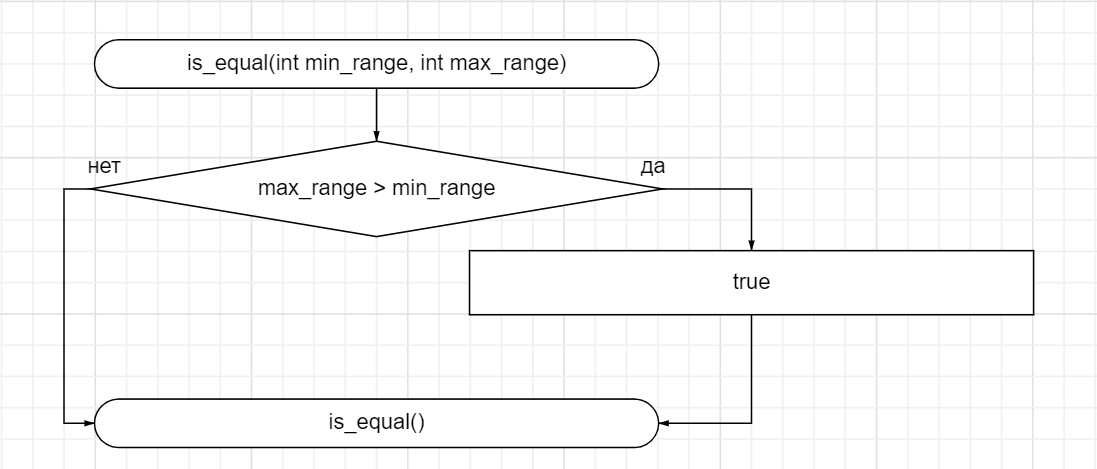


Рисунок 23 – Блок-схема используемых функций

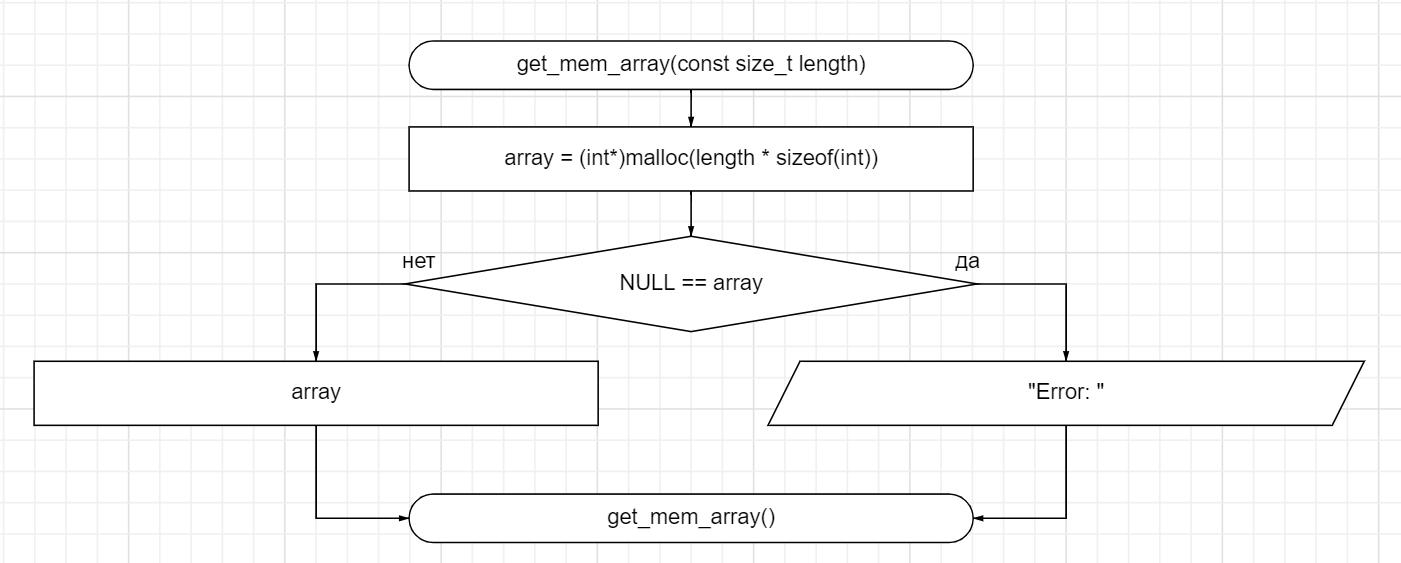
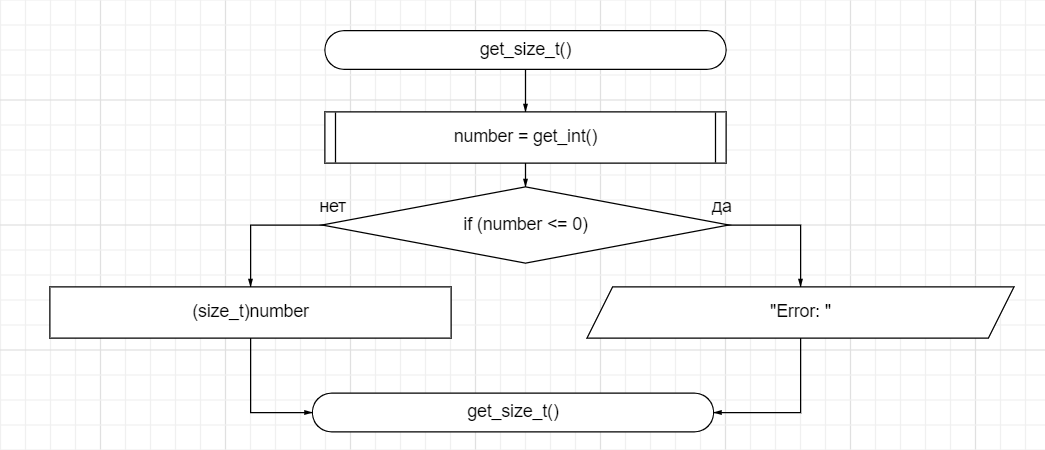


Рисунок 24 – Блок-схема используемых функций  
Рисунок 25 – Блок-схема используемых функций

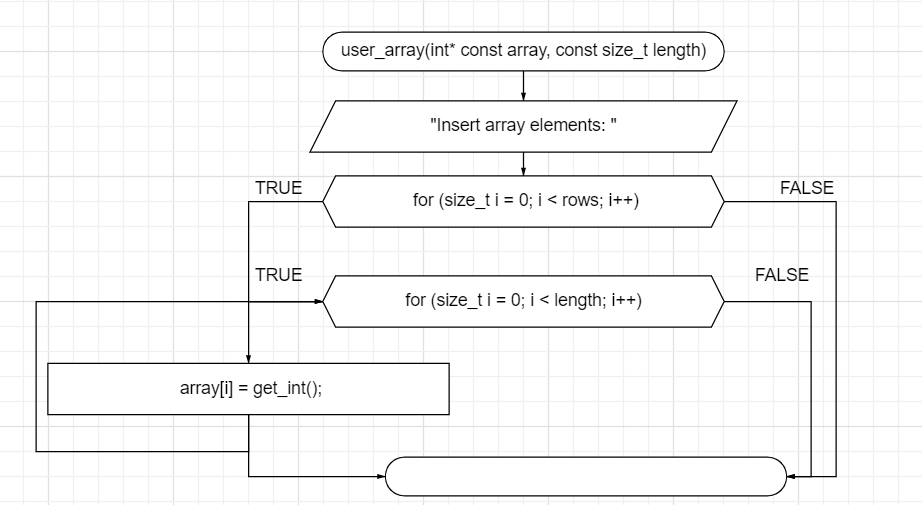
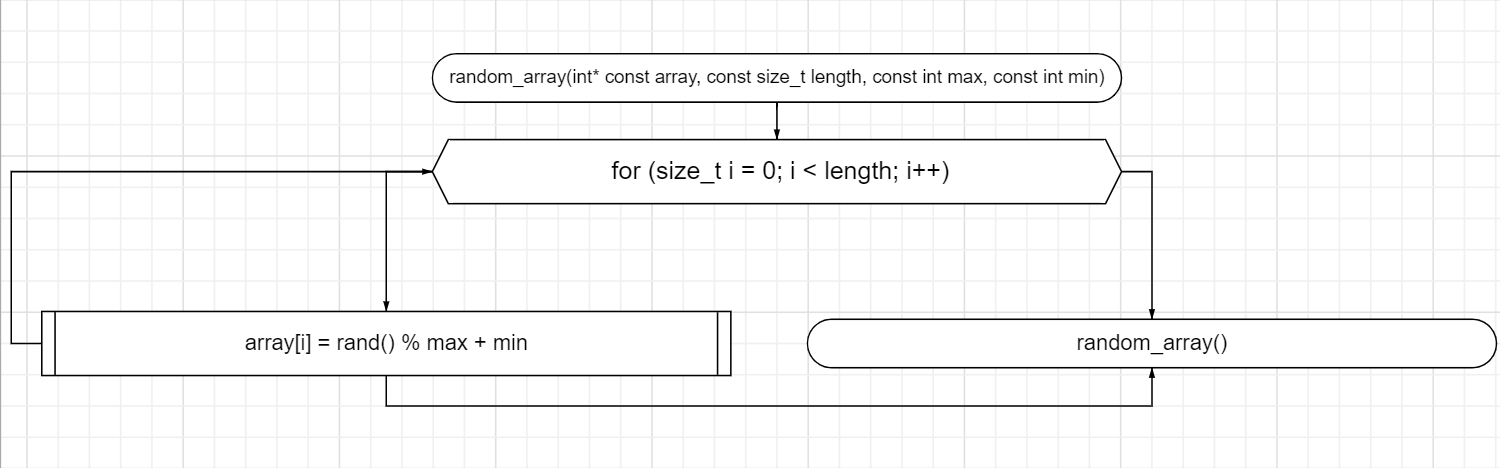
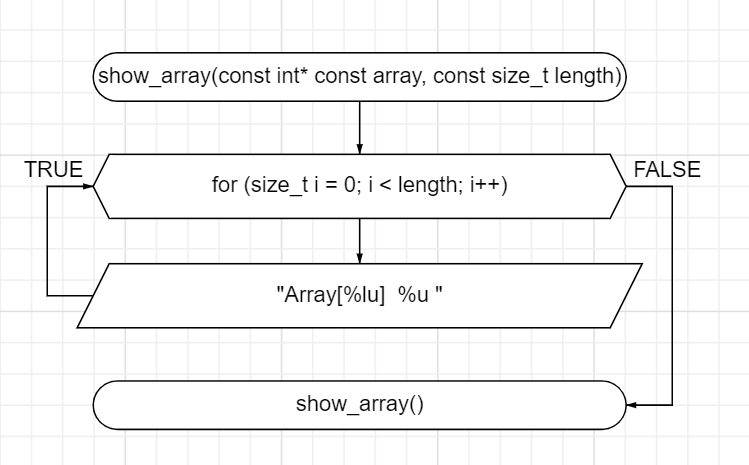
   
Рисунок 26 – Блок-схема используемых функций  


Рисунок 27 – Блок-схема используемых функций  
   
Рисунок 28 – Блок-схема используемых функций

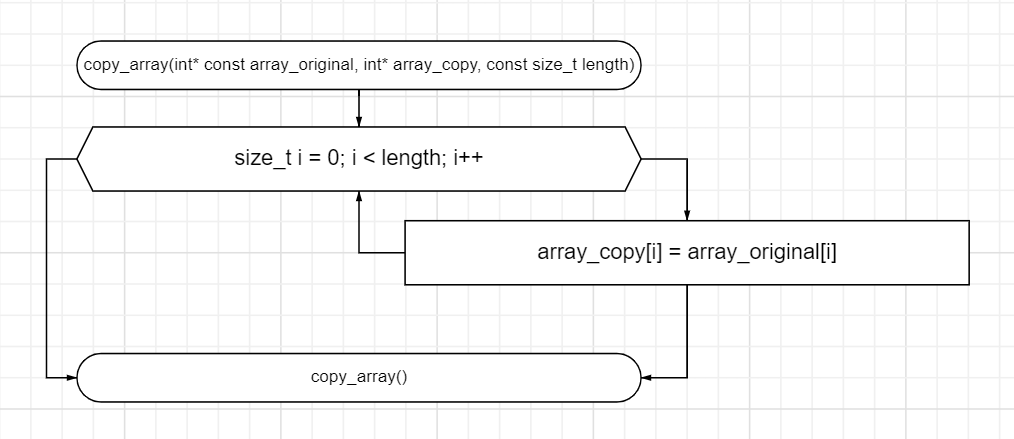
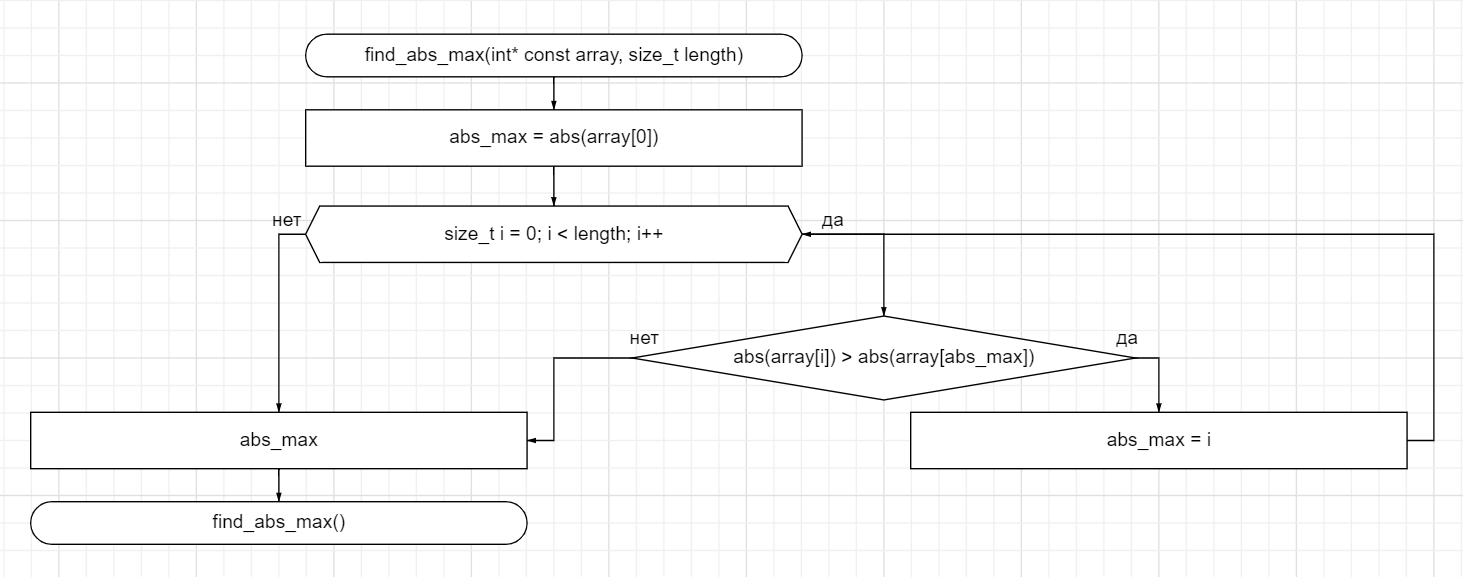
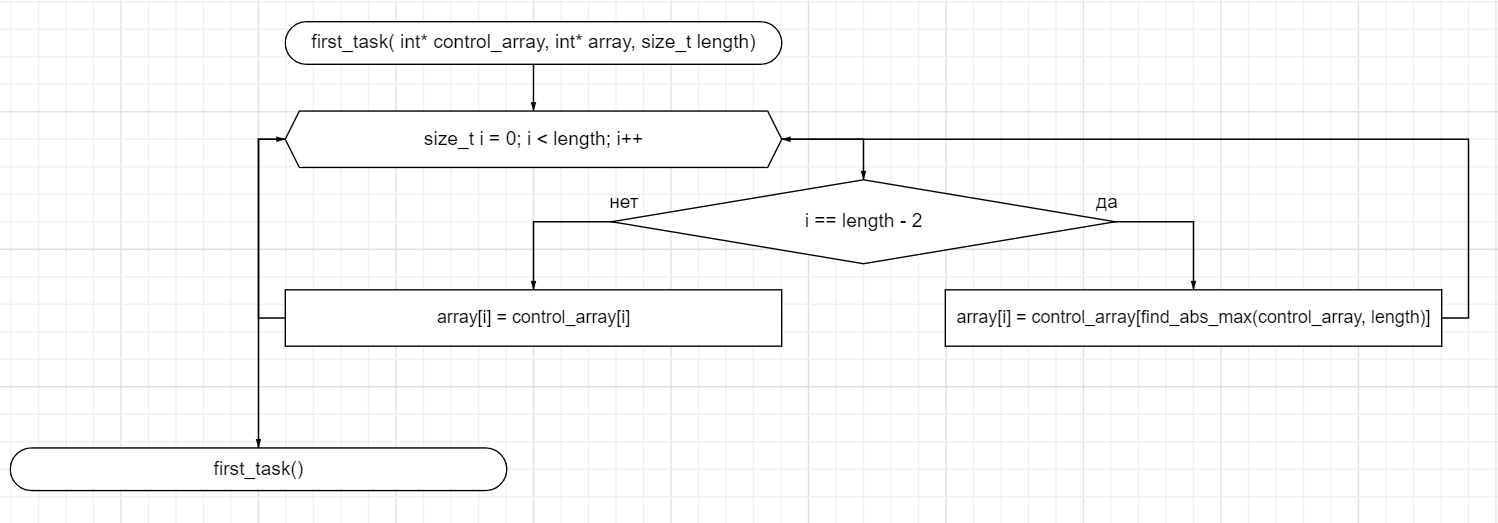
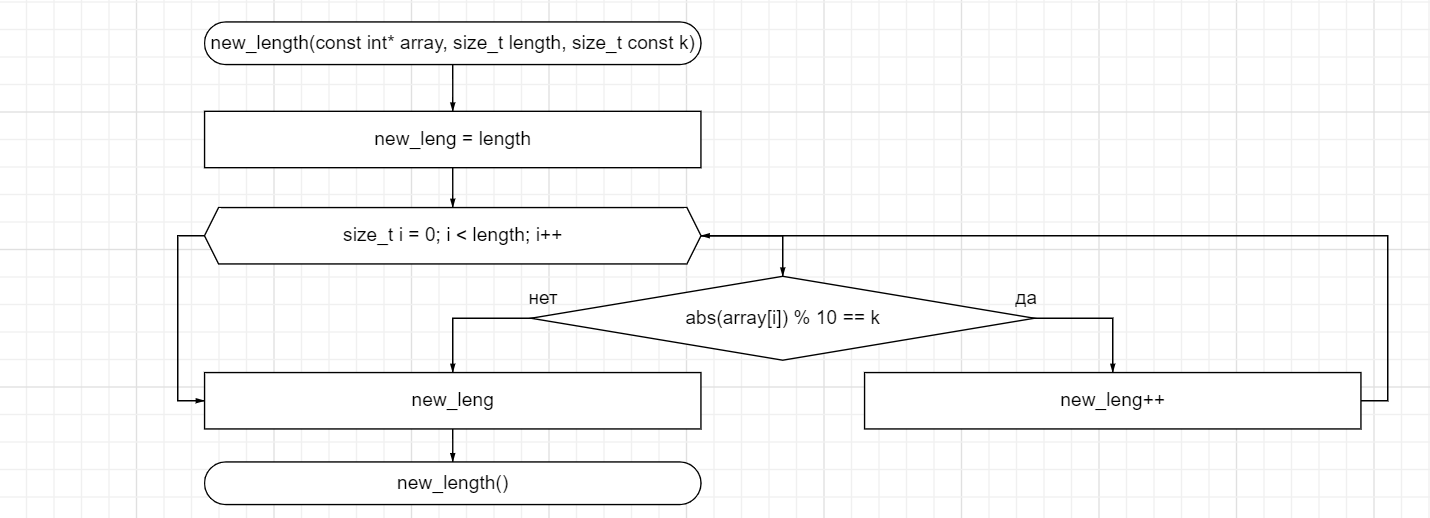
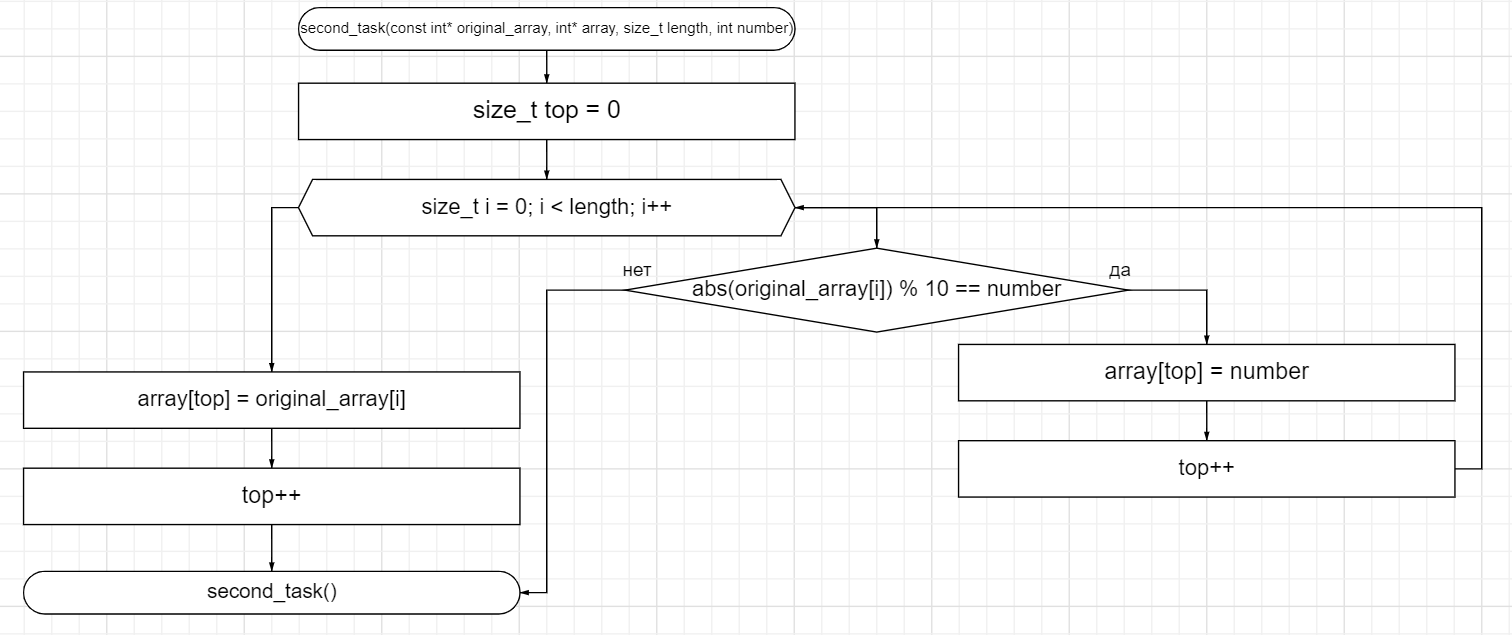
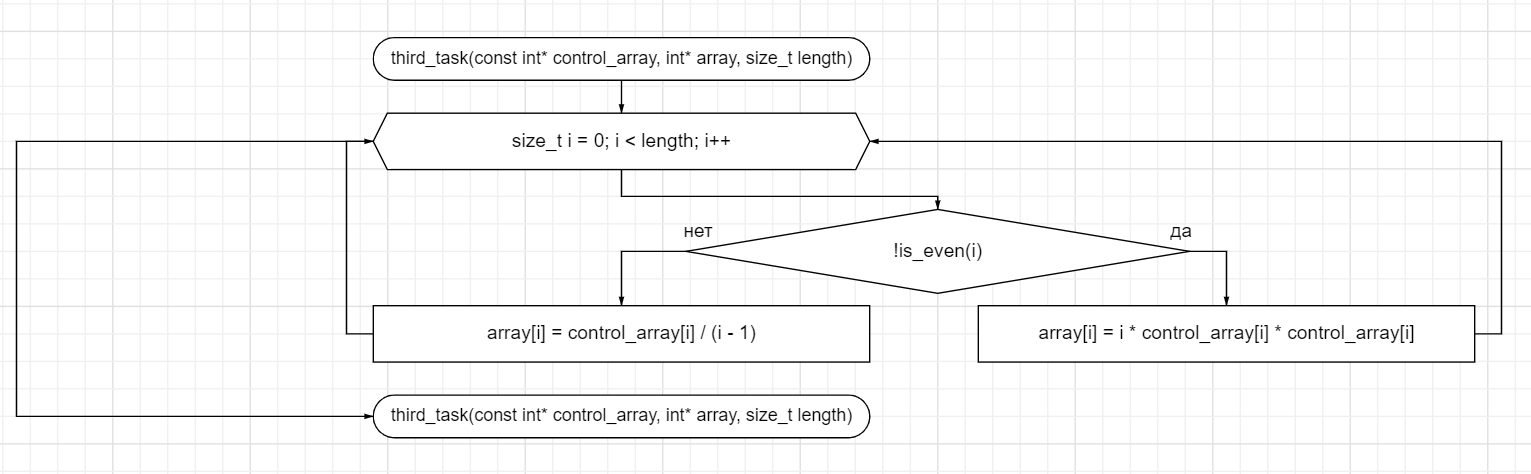
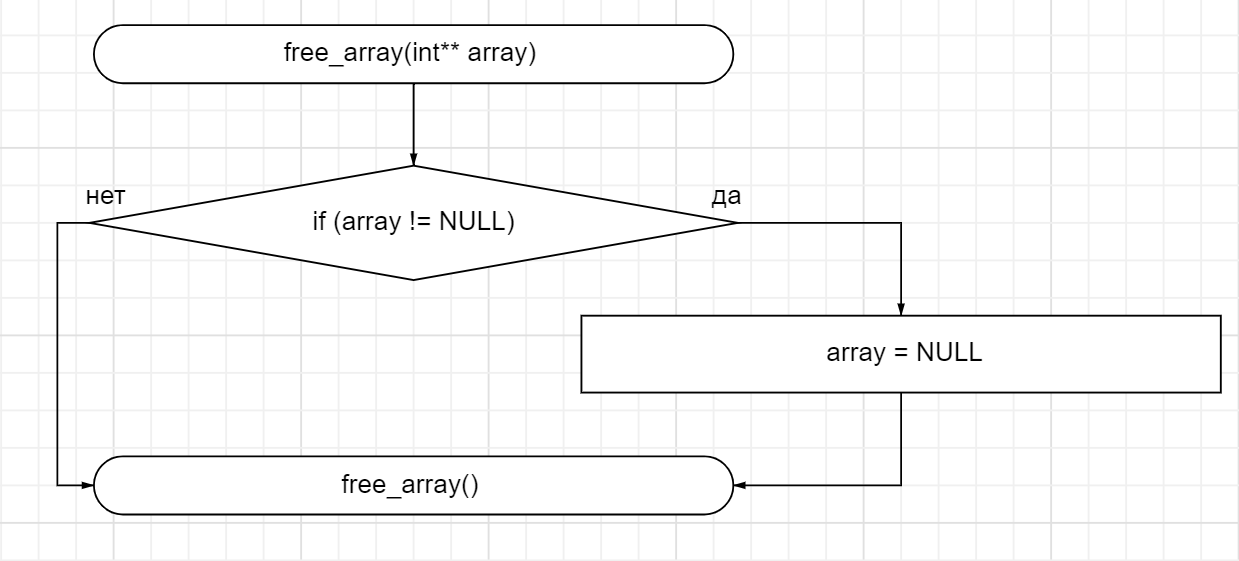


Рисунок 29 – Блок-схема используемых функций  
   
Рисунок 30 – Блок-схема используемых функций  
   
Рисунок 31 – Блок-схема используемых функций  
  
Рисунок 32 – Блок-схема используемых функций  
  
 Рисунок 33 – Блок-схема используемых функций  
  
Рисунок 34 – Блок-схема используемых функций  
  
  
Рисунок 35 – Блок-схема используемых функций

2.3 Текст программы на языке C

#include <stdio.h>

#include <stdlib.h>

#include <errno.h>

#include <stdbool.h>

/\*\*

\* @brief Функция присваивает целочисленное значение переменной

\* @return число

\*/

int get\_int();

/\*\*

\* @brief Функция определяет выполняется ли условие максимальное больше минимального

\* @param min\_range - значение минимального диапозона, вводимого пользователем

\* @param max\_range - значение максимального диапозона, вводимого пользователем

\* @return true, если условие выполняется, иначе объясняет что Введенные значения не удовлетворяют условию

\*/

bool is\_equal\_range(int min\_range, int max\_range);

/\*\*

\* @brief Функция присваивает переменной целочисленное значение и проверяет его на положителность

\* @return number положительное число

\*/

size\_t get\_size\_t();

/\*\*

\* @brief структура хранит константы, указывающие выбор пользователя заполнить массив

\* @brief user\_choice хранит значение, вызывающее ввод массива вручную

\* @brief random\_choice хранит значение, вызывающее ввод массива случайными числами

\*/

enum Choices

{

user\_choice = 1,

random\_choice = 2,

};

/\*\*

\* @brief Функция выделяет память под массив

\* @param length длина массива

\* @return array указатель на пустой массив

\*/

int\* get\_array(const size\_t length);

/\*\*

\* @brief фукнция считает новую длину под масиив из пункта 2

\* @param array исходный массив

\* @param length длина исходного массива

\* @param k введенное число

\*/

size\_t new\_length(const int\* array, size\_t length, size\_t const k);

/\*\*

\* @brief Функция заполняет массив, элементы вводит пользователь

\* @param array указатель на заполняемый массив

\* @param length длина массива

\*/

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

/\*\*

\* @brief Функция заполняет массив рандомными числами в диапазоне [-15:15]

\* @param array указатель на заполняемый массив

\* @param length длина массива

\*/

void random\_array(int\* const array, const size\_t length, const int max\_range, const int min\_range);

/\*\*

\* @brief Функция, показывающая массив поэлементно

\* @param array ссылка на показываемый массив

\* @param length длина массива

\*/

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

/\*\*

\* @brief Заменяет предпоследний элемент массива на максимальный по модулю.

\* @param array ссылка на массив

\* @param length размер массива

\* @return new\_array новый массив, соответствующий этим требованиям

\*/

void first\_task( int\* control\_array, int\* array, size\_t length);

/\*\*

\* @brief ЗАПОЛНЯЕТ массив по условию пункта 2

\* @param original\_array исходный массив

\* @param array пустой новый массив с нужной размерностью

\* @param length длина исходного массива

\* @param number введенное число

\*/

void second\_task(const int\* original\_array, int\* array, size\_t length, int number);

/\*\*

\* @brief Функция проверяет на четность целое число

\* @param number число

\* @return 0 если четное число

\* @return 1 если нечетное число

\*/

bool is\_even(int number);

/\*\*

\* @brief Заменяет в массиве элементы по соответствующим условиям

\* @param array ссылка на массив

\* @param length размерность массива

\* @return new\_array новый массив, соответствующий этим требованиям

\*/

void third\_task(const int\* control\_array, int\* array, size\_t length);

/\*\*

\* @brief Функция, освобождающая массив

\* @param array указатель на массив

\*/

void free\_array(int\*\* array);

/\*\*

\* @brief Точка входа в программу

\* @return 0 Программа исправна

\*/

int main()

{

puts("Enter the min\_range: ");

const int min\_range = get\_int();

puts("Enter the max\_range: ");

const int max\_range = get\_int();

is\_equal(min\_range, max\_range);

puts("insert a length of array\n");

size\_t length = get\_size\_t();

printf("if you fill array by youself, press %d, if you fill array by random numbers, press %d\n", (enum Choices)(user\_array), (enum Choices)(random\_array));

int choice = get\_int();

int\* mas = get\_array(length);

switch ((enum Choices)choice)

{

case random\_choice:

random\_array(mas, length, min\_range, max\_range);

break;

case user\_choice:

user\_array(mas, length);

break;

default:

puts("Insert a valid choice!\n");

return 1;

break;

}

puts("Array\n");

show\_array(mas, length);

int\* task\_1\_array = get\_array(length);

first\_task(mas, task\_1\_array, length);

puts("First task\n");

show\_array(task\_1\_array, length);

free\_array(&task\_1\_array);

puts("Insert a pozitive, single digit number\n");

int k = get\_int();

if (k < 0 || k > 10)

{

puts("Insert a valid number!\n");

return 1;

}

int\* task\_2\_1 = get\_array(new\_length(mas, length, k));

second\_task(mas, task\_2\_1, length, k);

puts("Second task\n");

show\_array(task\_2\_1, new\_length(mas, length, k));

free\_array(&task\_2\_1);

int\* task\_3\_array = get\_array(length);

third\_task(mas, task\_3\_array, length);

puts("Third task\n");

show\_array(task\_3\_array, length);

free\_array(&task\_3\_array);

free\_array(&mas);

return 0;

}

int get\_int()

{

int number;

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

{

puts("Inserted a wrong value\n");

abort();

}

return number;

}

bool is\_equal(int min\_range, int max\_range)

{

if(max\_range > min\_range)

{

return true;

}

abort();

}

size\_t get\_size\_t()

{

int number = get\_int();

if (number <= 0)

{

errno = ERANGE;

perror("Error: \n");

abort();

}

return (size\_t)number;

}

int\* get\_mem\_array(const size\_t length)

{

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

if (NULL == array)

{

errno = ENOMEM;

perror("Error: ");

abort();

}

return array;

}

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

{

puts("Insert array elements:\n");

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

{

array[i] = get\_int();

}

}

void random\_array(int\* const array, const size\_t length, const int max\_range, const int min\_range)

{

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

{

array[i] = rand() % max\_range + min\_range;

}

}

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

{

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

{

printf("Element[%ld] %d\n", i, array[i]);

}

}

void copy\_array(int\* const array\_original, int\* array\_copy, const size\_t length)

{

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

{

array\_copy[i] = array\_original[i];

}

}

int find\_abs\_max(int\* const array, size\_t length)

{

int abs\_max = abs(array[0]);

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

{

if (abs(array[i]) > abs(array[abs\_max]))

{

abs\_max = i;

}

}

return abs\_max;

}

bool is\_even(int number)

{

return number % 2 == 0;

}

void first\_task( int\* control\_array, int\* array, size\_t length)

{

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

{

if (i == length - 2)

{

array[i] = control\_array[find\_abs\_max(control\_array, length)];

}

else

{

array[i] = control\_array[i];

}

}

}

size\_t new\_length(const int\* array, size\_t length, size\_t const k)

{

size\_t new\_leng = length;

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

{

if (abs(array[i]) % 10 == k)

{

new\_leng++;

}

}

return new\_leng;

}

void second\_task(const int\* original\_array, int\* array, size\_t length, int number)

{

size\_t top = 0;

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

{

if (abs(original\_array[i]) % 10 == number)

{

array[top] = number;

top++;

}

array[top] = original\_array[i];

top++;

}

}

void third\_task(const int\* control\_array, int\* array, size\_t length)

{

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

{

if (!is\_even(i))

{

array[i] = i \* control\_array[i] \* control\_array[i];

}

else

{

array[i] = control\_array[i] / (i - 1);

}

}

}

void free\_array(int\*\* array)

{

if (array != NULL)

{

free(array);

array = NULL;

}

}

2.4 Результаты выполнения программы

Результаты выполнения программы представлены ниже (Рисунок 36-37).

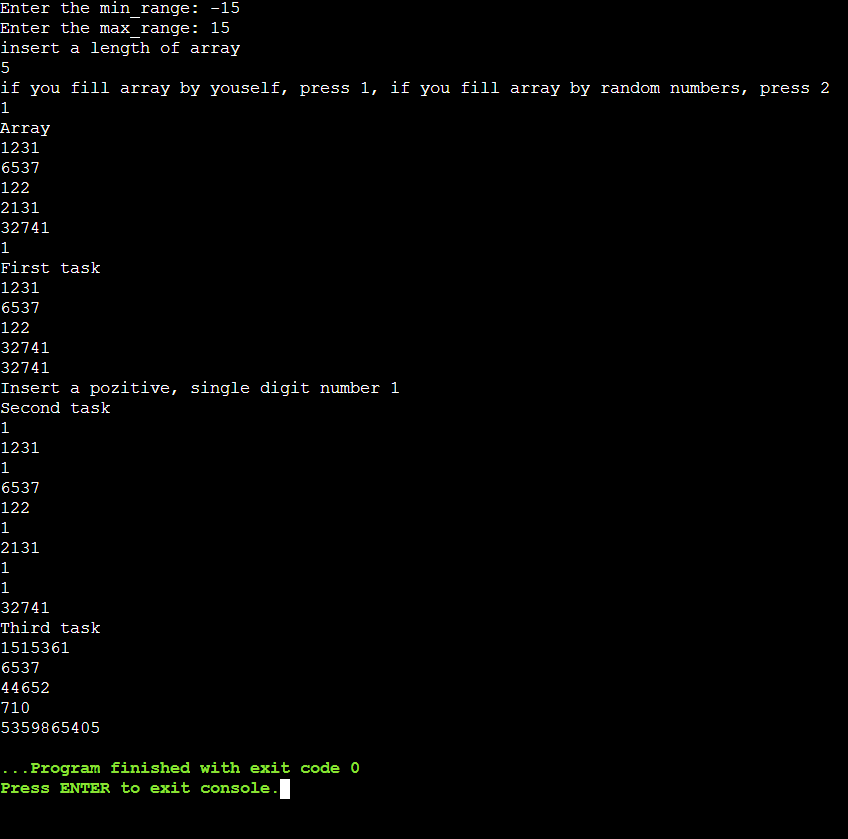


Рисунок 36– Результаты выполнения программы

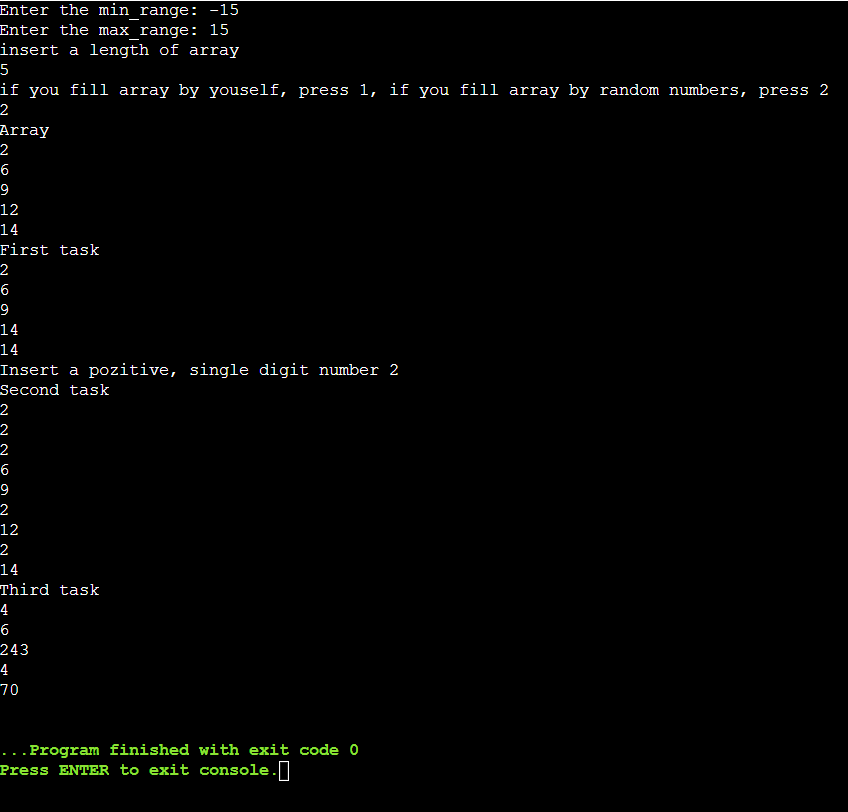


Рисунок 37– Результаты выполнения программы

2.5 Выполнение тестовых примеров

2.6 Отметка о выполнении задания в веб-хостинге системы контроля версий

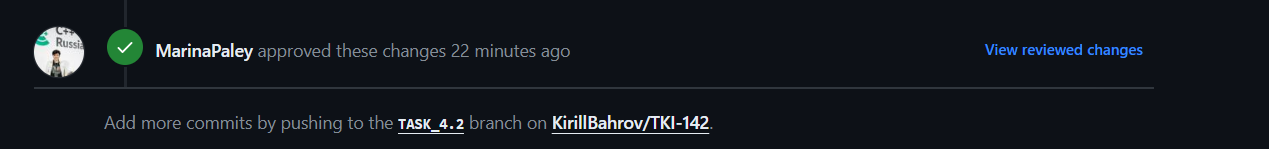
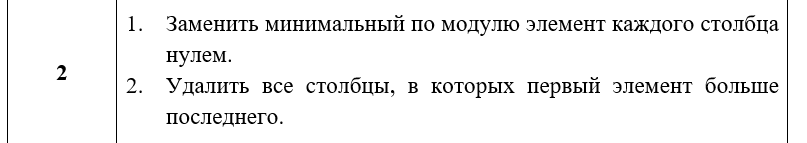


Рисунок 38 – Результат выполнения задания в веб хостинге контроля версий

РЕШЕНИЕ ЗАДАЧИ 4.3  
3.1 Формулировка задания

Создать многомерный массив n ˣ m 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.

Таблица 3 – Исходные данные



3.2 Блок-схема алгоритма

Блок-схема основного алгоритма представлена ниже (Рисунок 15). Блок-схемы используемых функций представлены ниже (Рисунки 16 - 19).

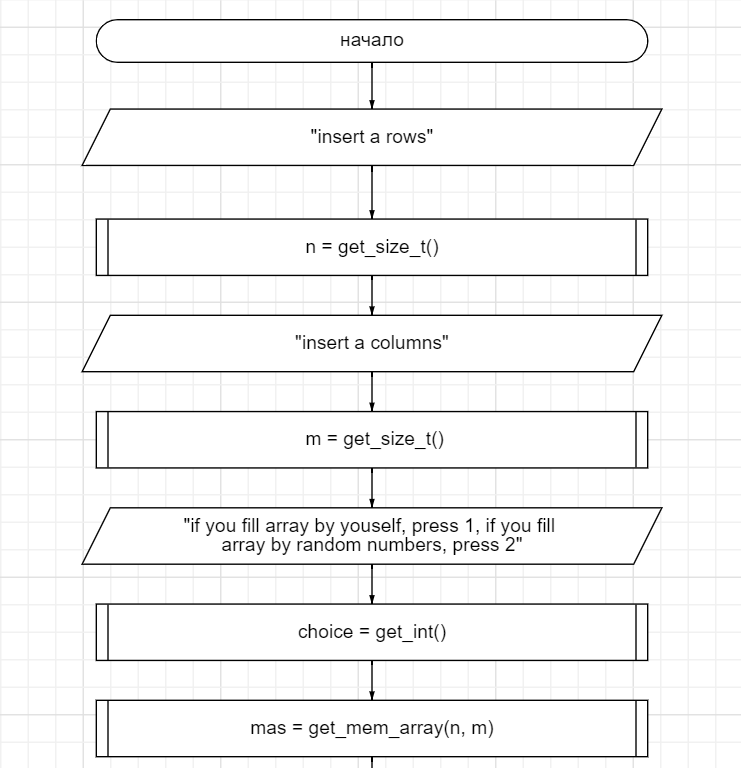
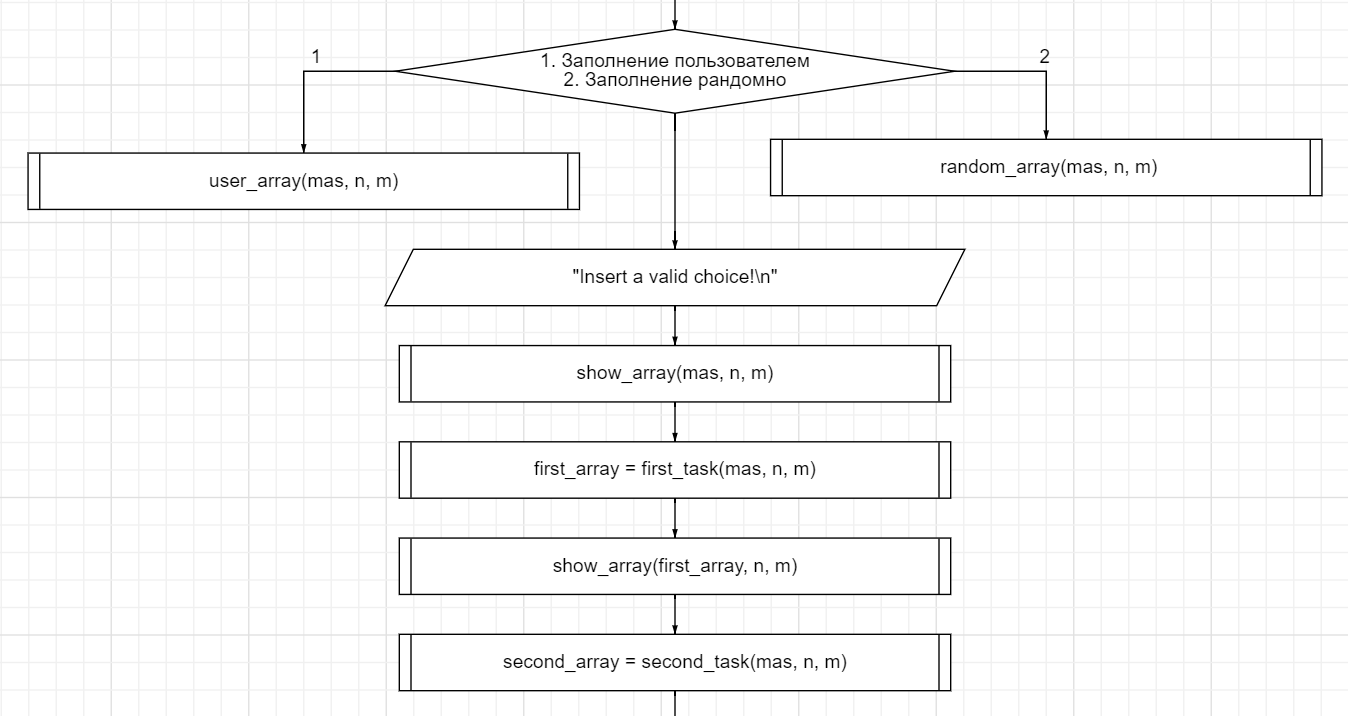


Рисунок 8 ­ Блок-схема основного алгоритма  
  
  
Рисунок 9 -  ­ Блок-схема основного алгоритма

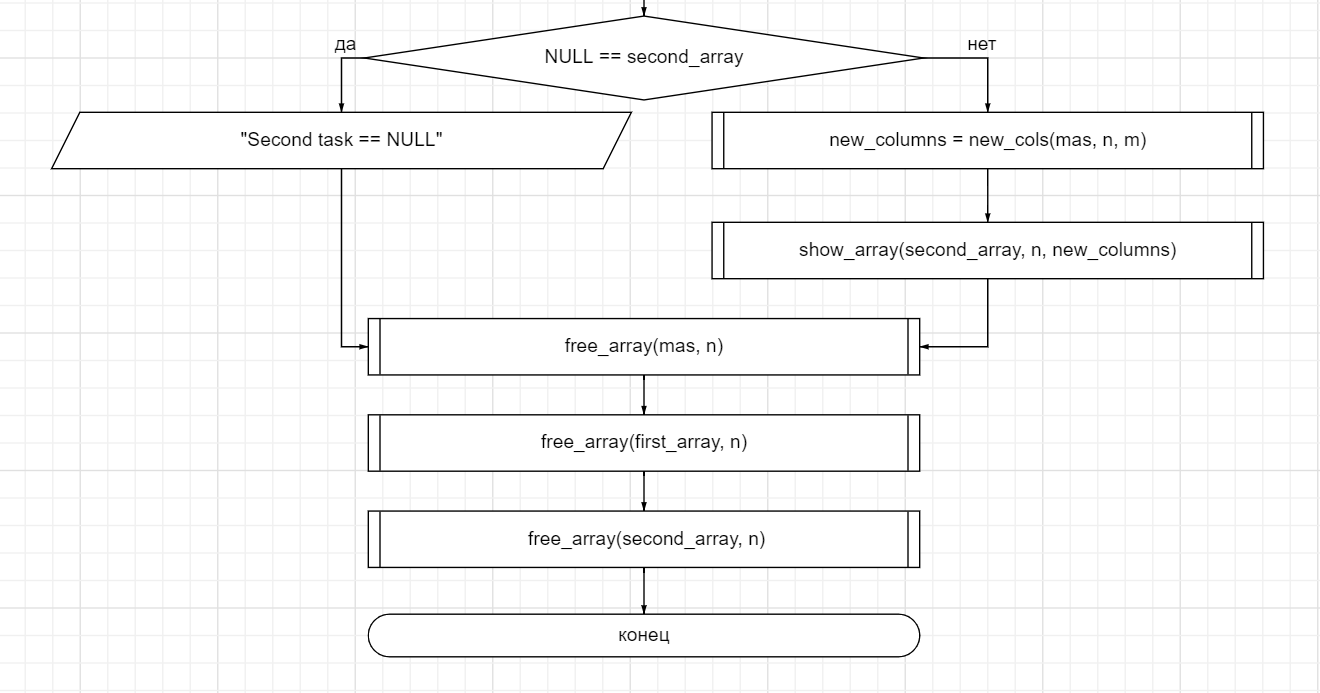
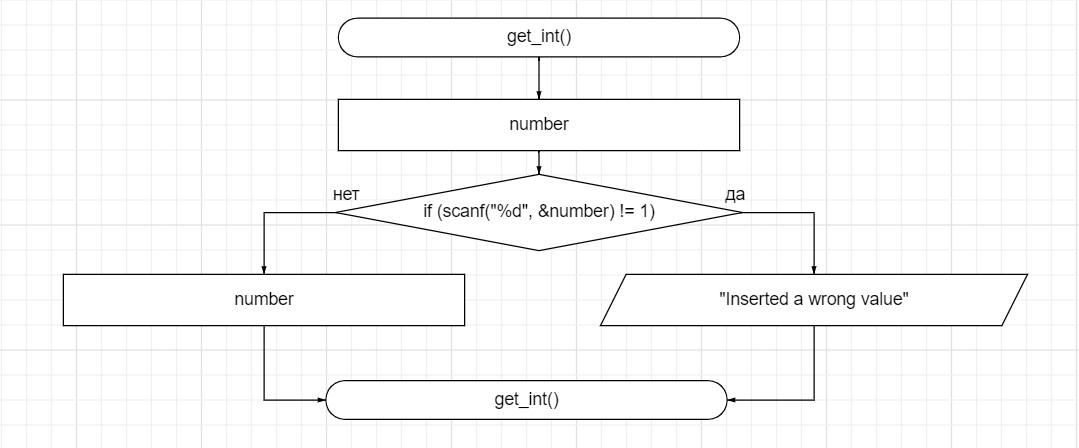
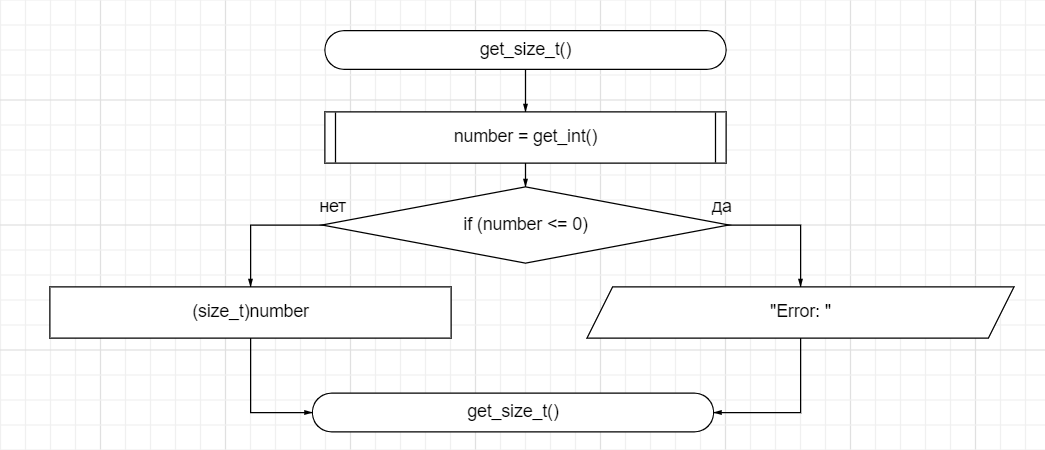
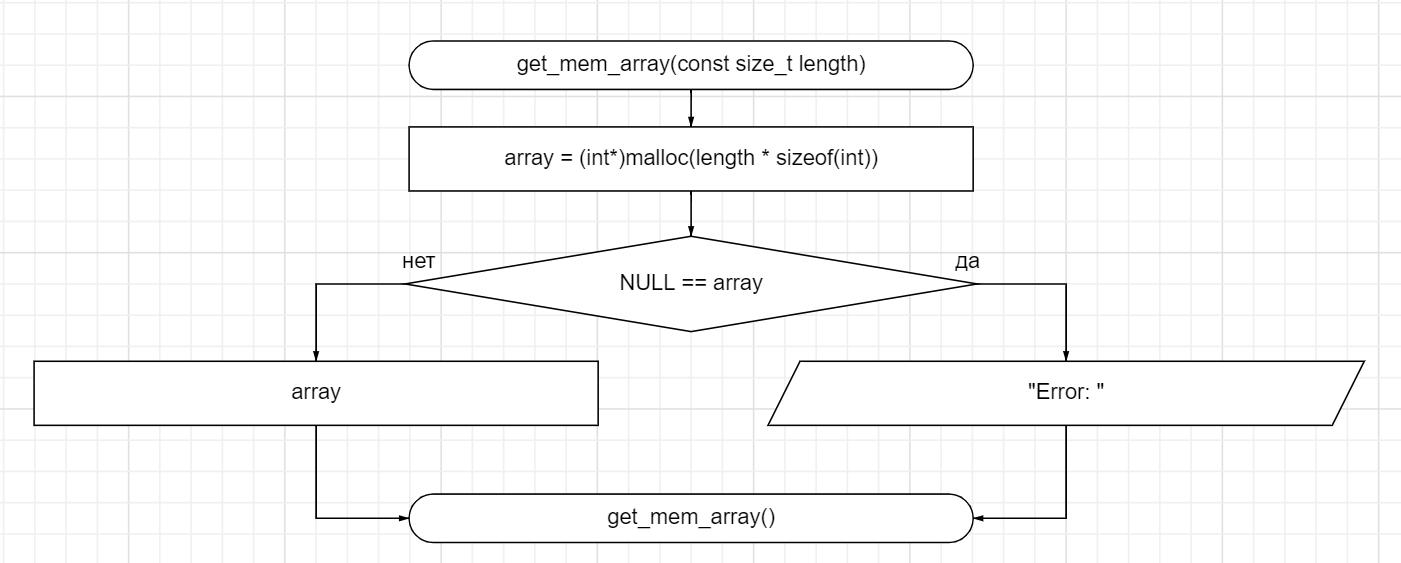
  
Рисунок 40 – Блок-схема используемых функций  
  
Рисунок 41 – Блок-схема используемых функций  
Рисунок 42 – Блок-схема используемых функций   


Рисунок 43 – Блок-схема используемых функций

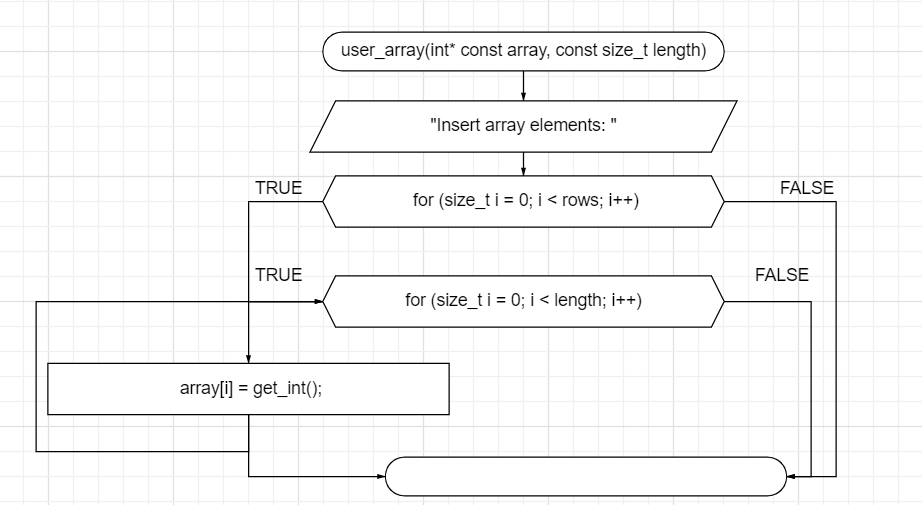
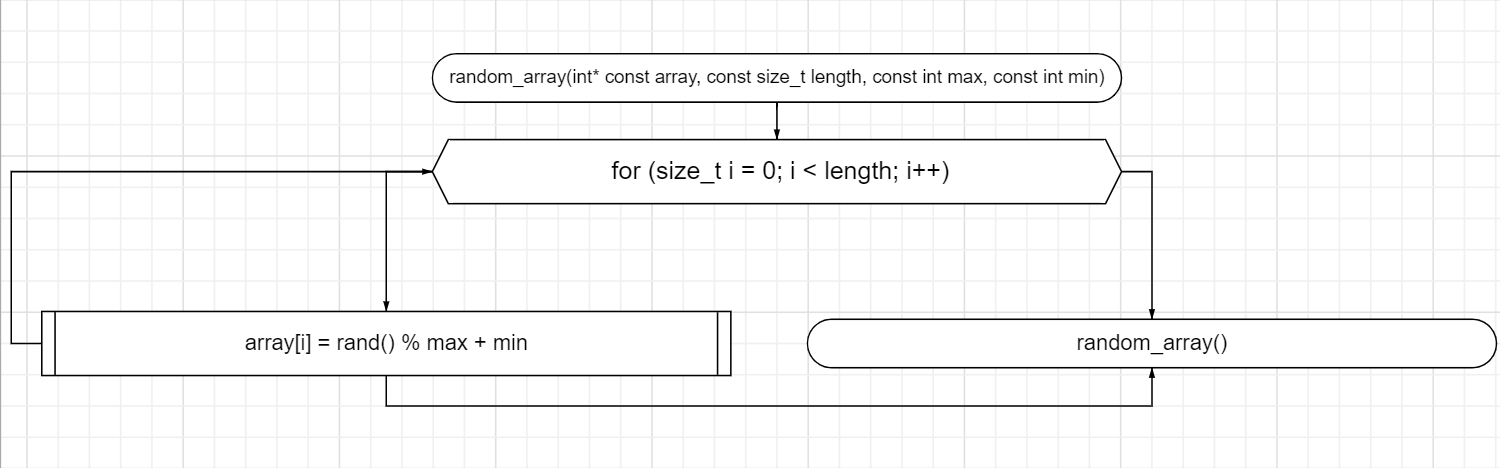
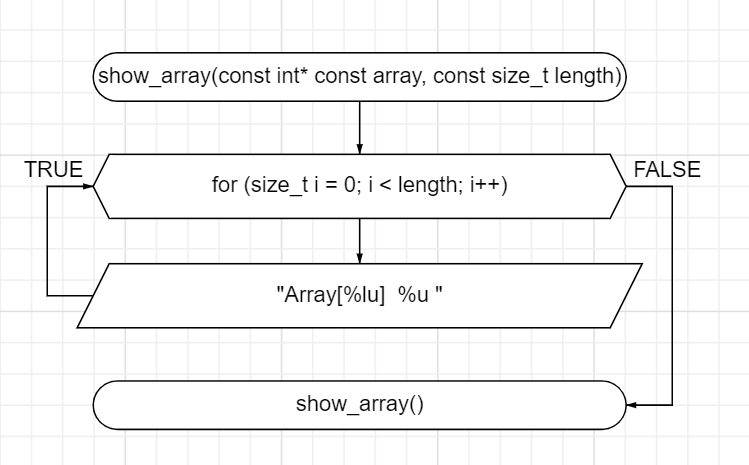
   
Рисунок 44 – Блок-схема используемых функций  


Рисунок 45 – Блок-схема используемых функций  
   
Рисунок 46 – Блок-схема используемых функций

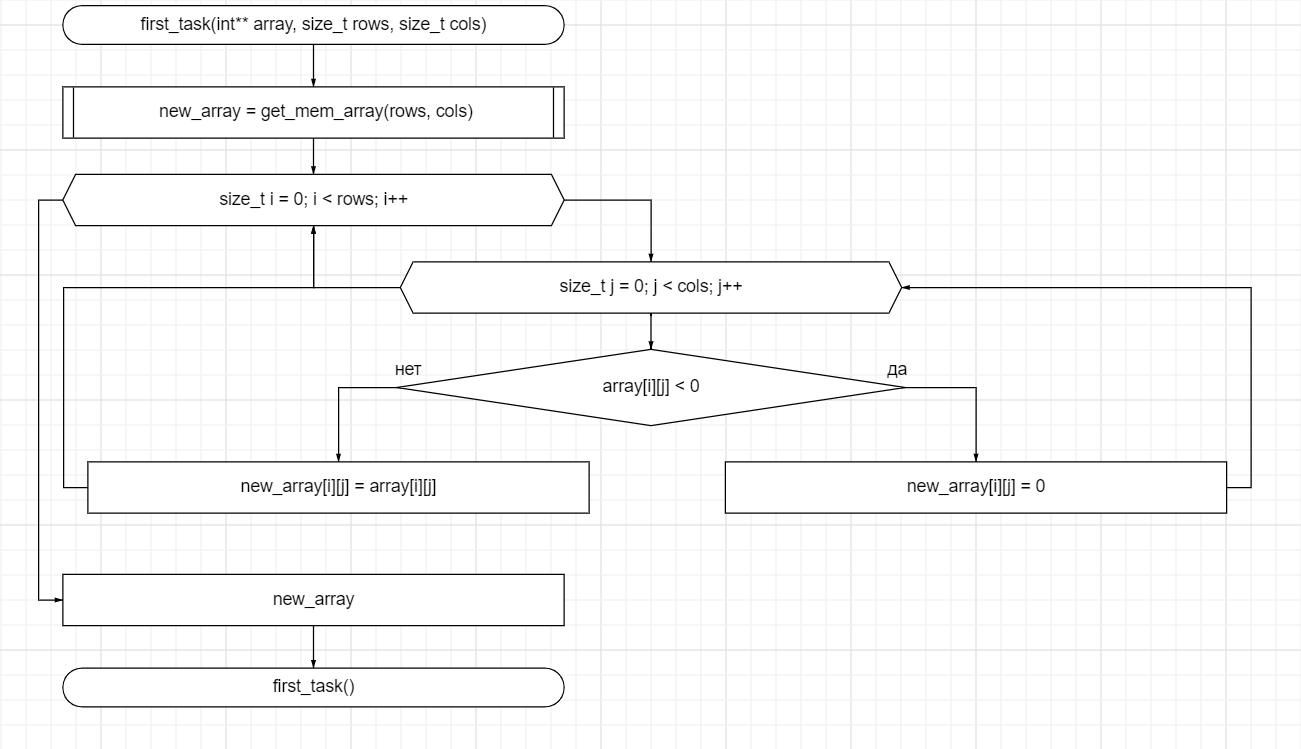
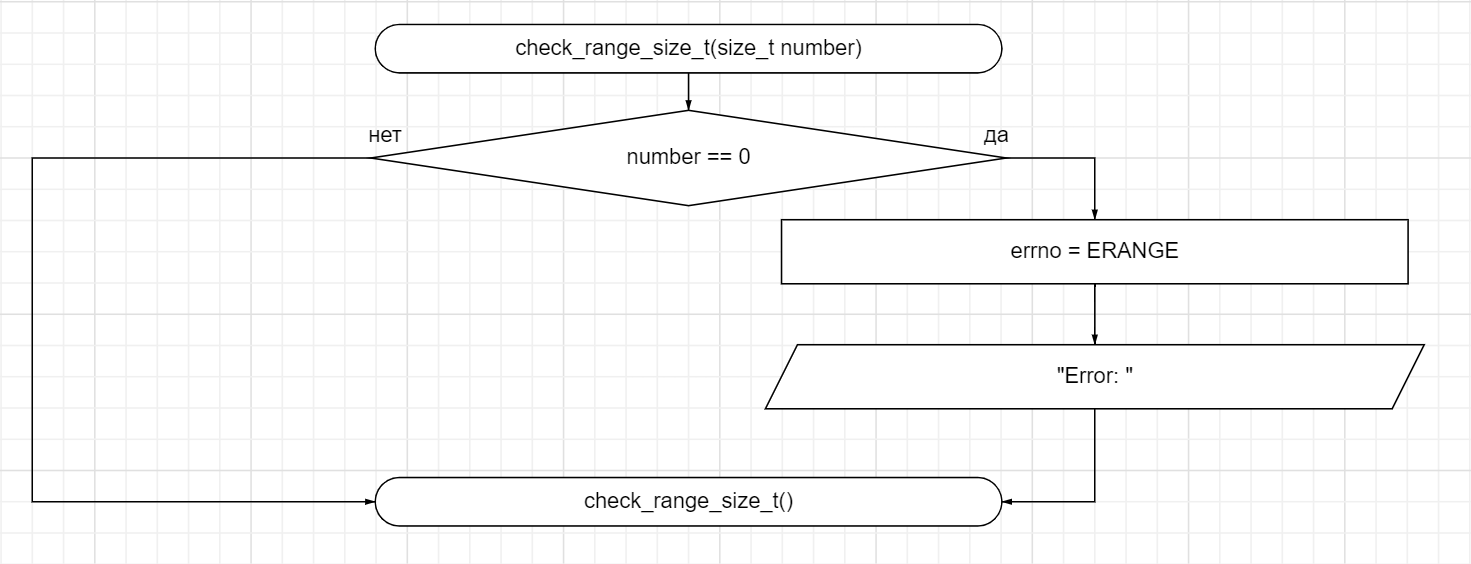
  
Рисунок 47 – Блок-схема используемых функций  


Рисунок 48 – Блок-схема используемых функций

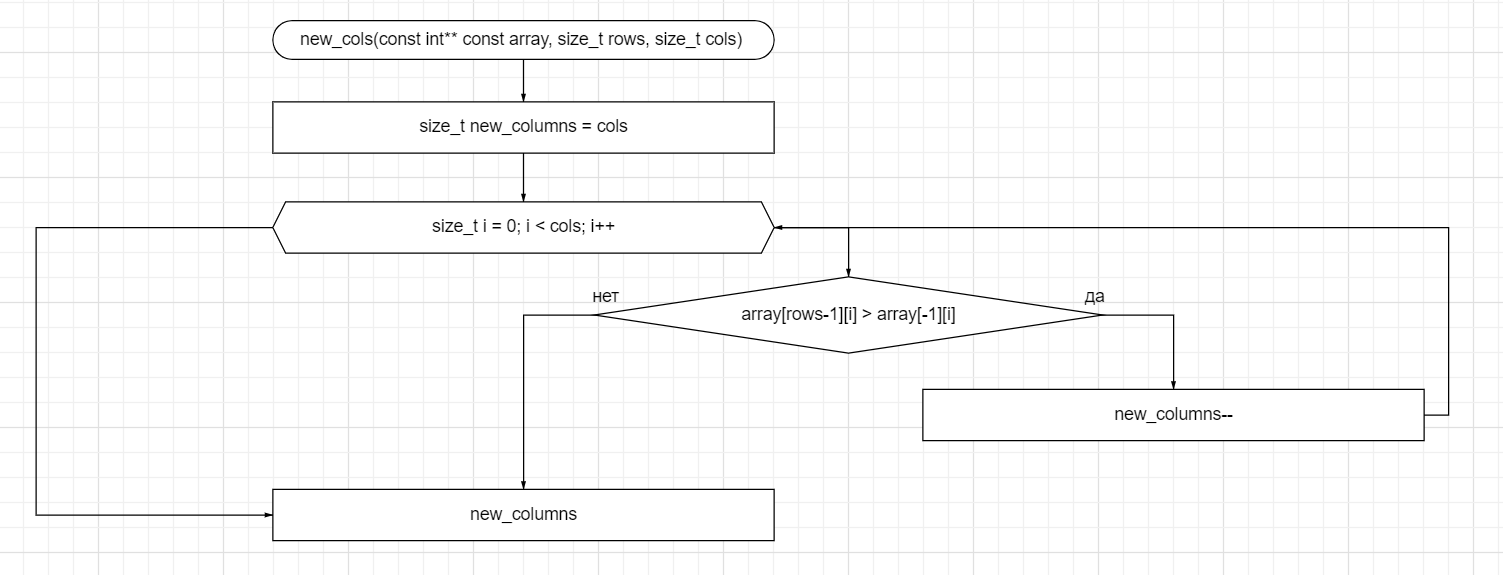


Рисунок 49 – Блок-схема используемых функций

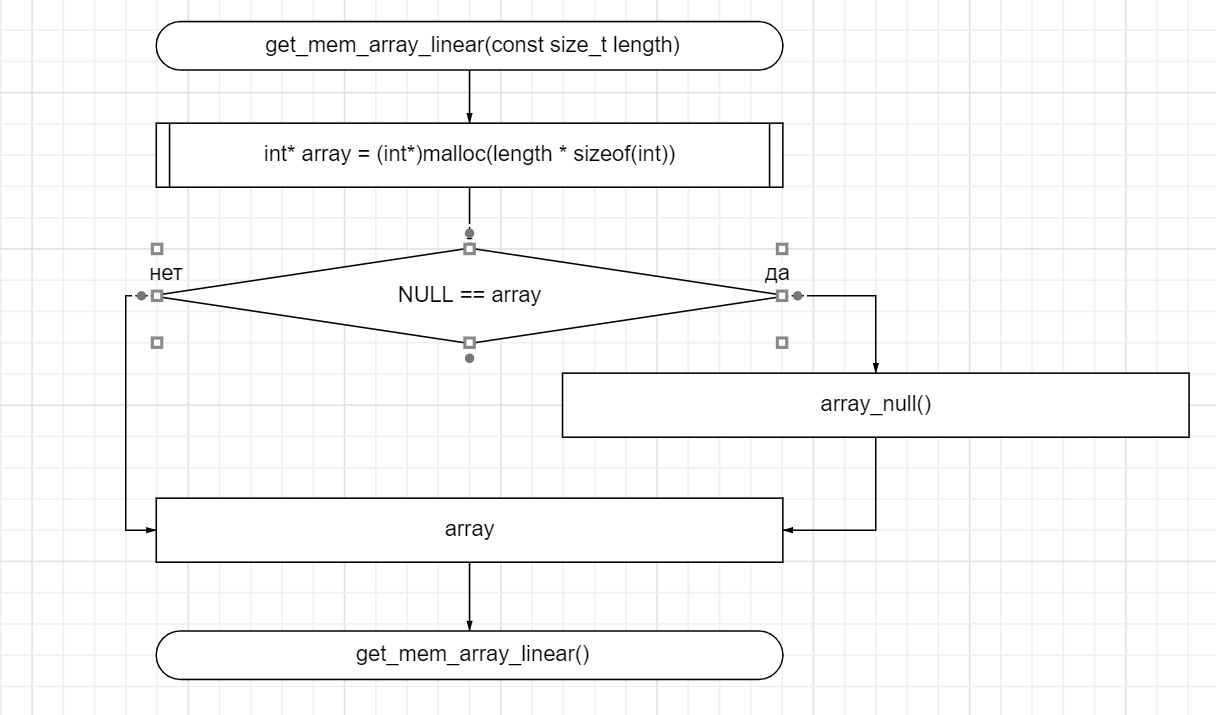


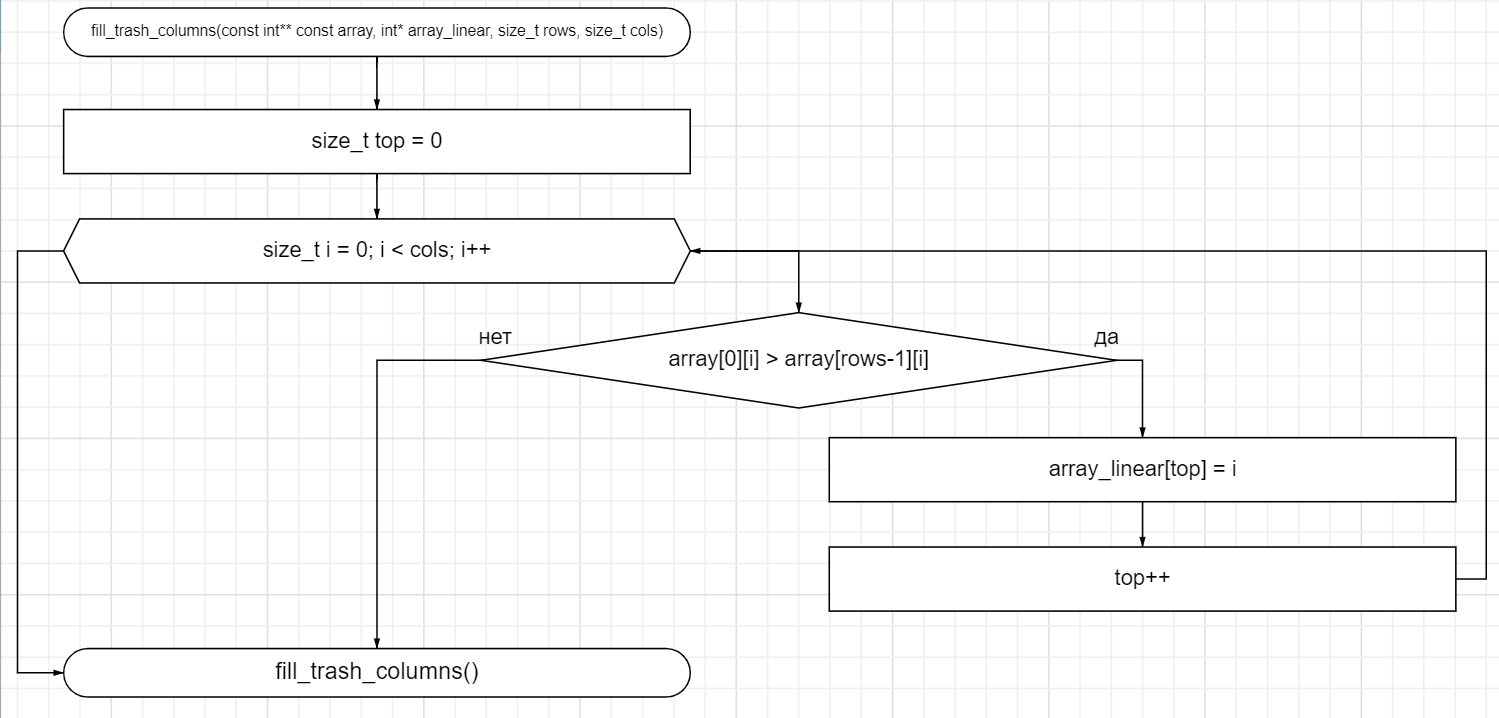
Рисунок 50 – Блок-схема используемых функций  


Рисунок 51 – Блок-схема используемых функций

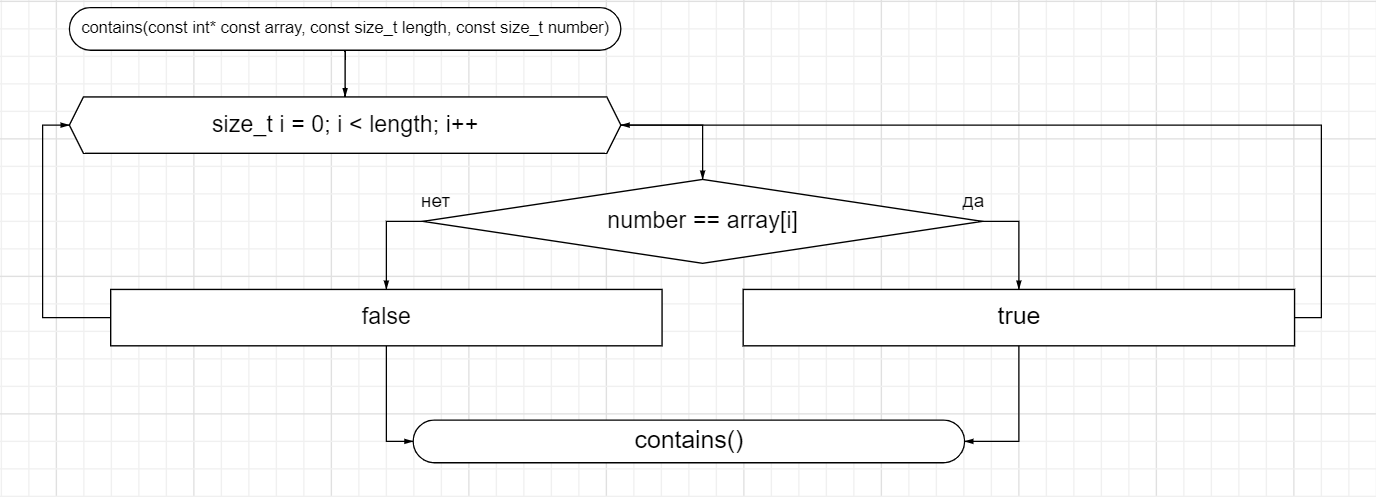


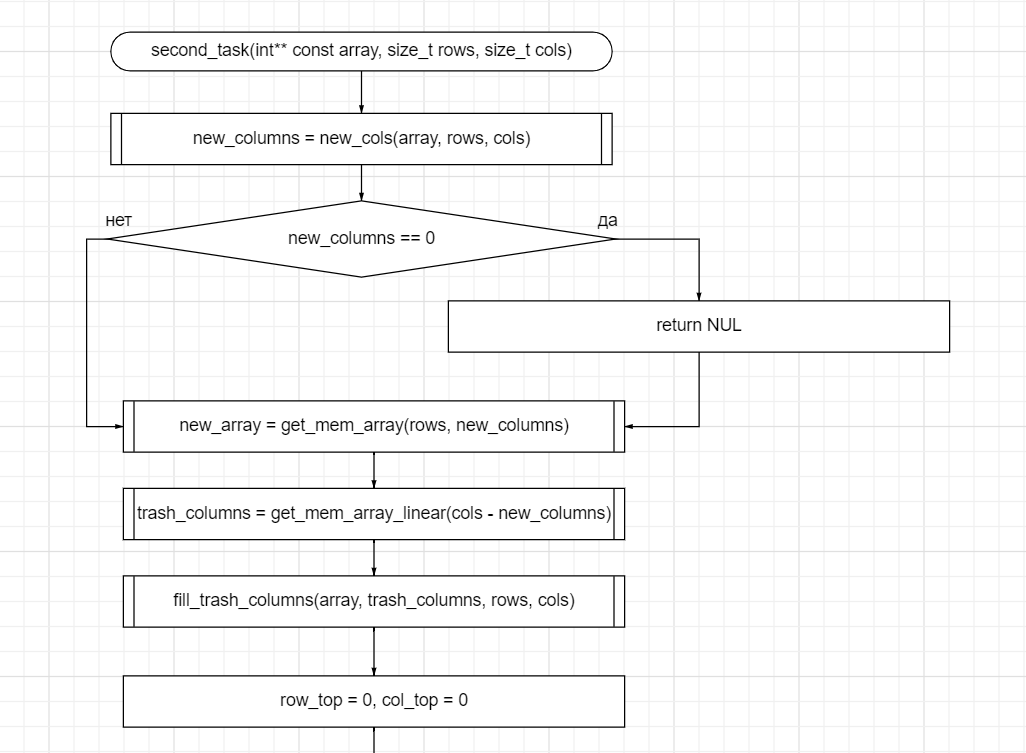
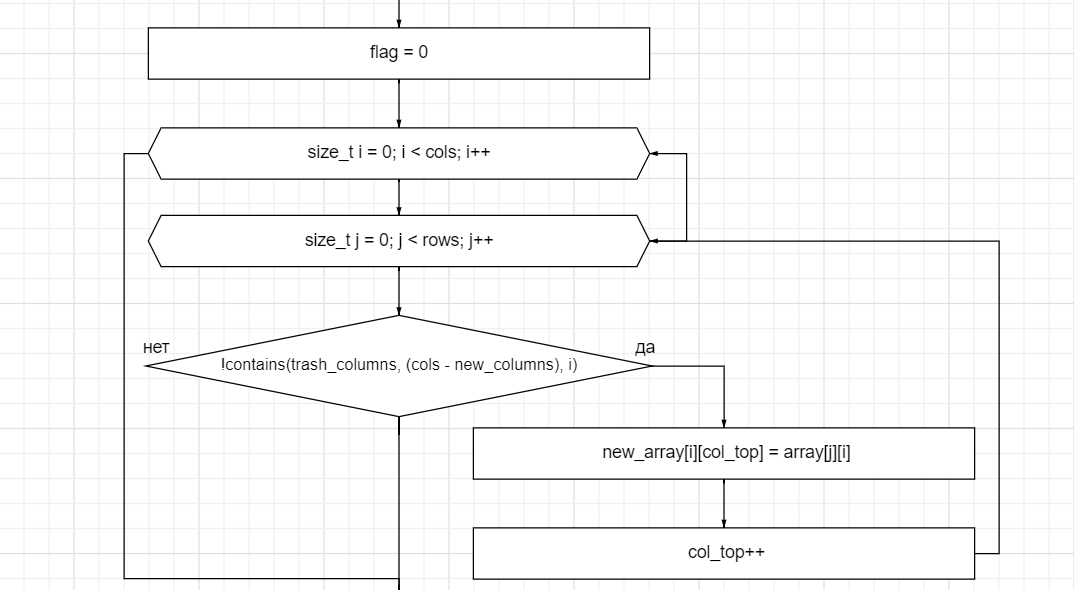
Рисунок 52 – Блок-схема используемых функций  


Рисунок 53 – Блок-схема используемых функций

  
Рисунок 54 – Блок-схема используемых функций

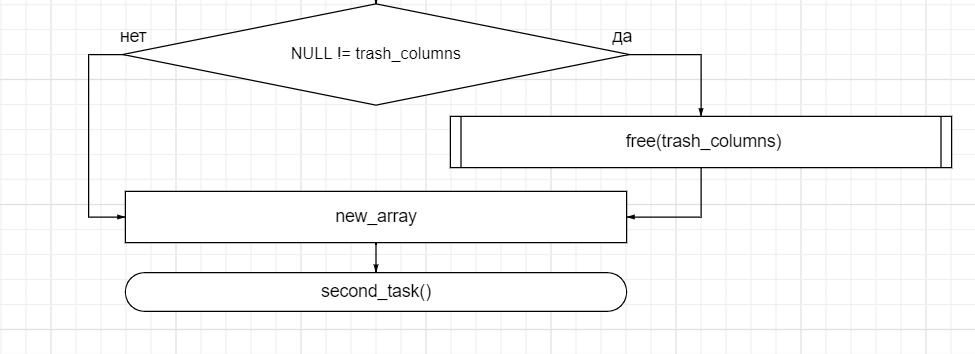


Рисунок 55 – Блок-схема используемых функций

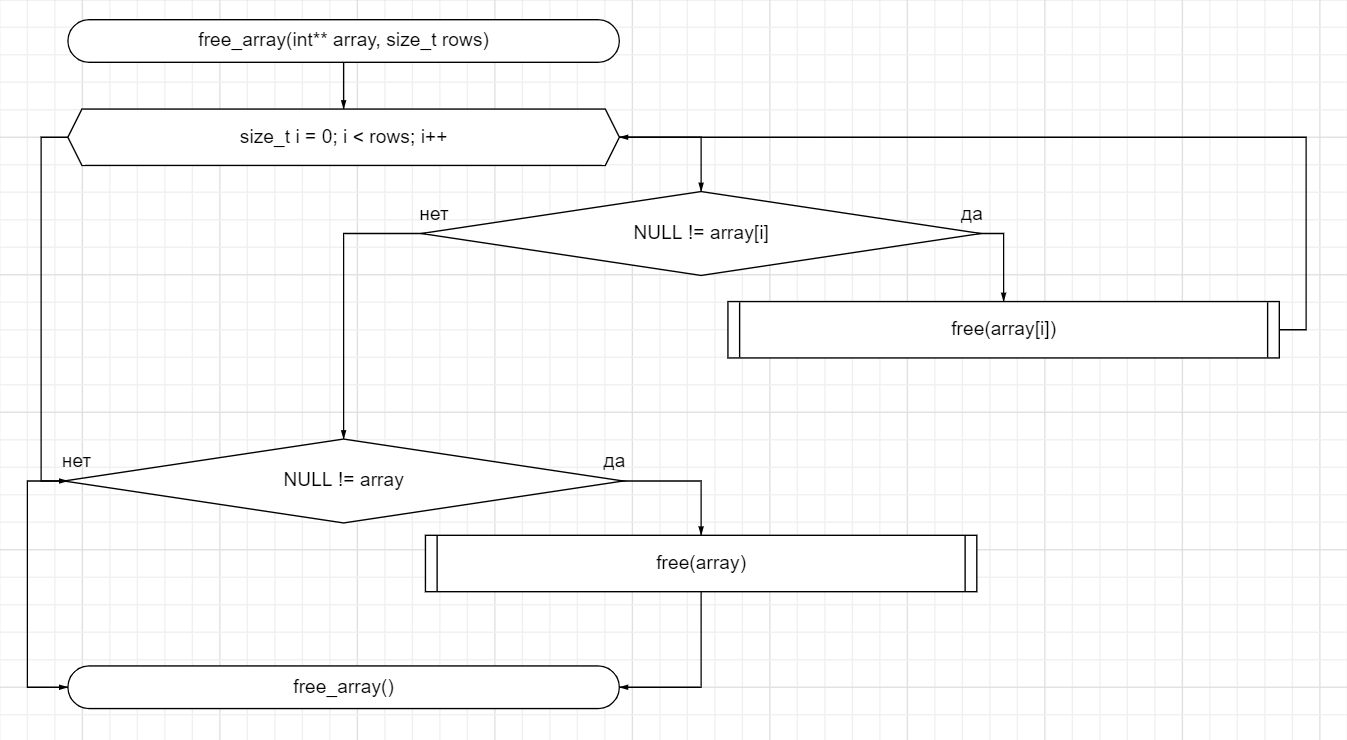


Рисунок 56 – Блок-схема используемых функций

3.3 Текст программы на языке C

#include <stdio.h>

#include <stdlib.h>

#include <errno.h>

#include <stdbool.h>

/\*\*

\* @brief Функция присваивает целочисленное значение переменной

\* @return целое число

\*/

int get\_int();

/\*\*

\* @brief Функция, прерывающая программу из-за отсутствия памяти для массивов

\*/

void array\_null();

/\*\*

\* @brief освобождает многомерный массив

\* @param array указатель на заполняемый массив

\* @param rows количество строк в массиве

\*/

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

/\*\*

\* @brief Функция присваивает переменной целочисленное значение и проверяет его на положителность

\* @return number положительное число

\*/

size\_t get\_size\_t();

/\*\*

\* @brief структура хранит константы, указывающие выбор пользователя заполнить массив

\* @brief user\_choice хранит значение, вызывающее ввод массива вручную

\* @brief random\_choice хранит значение, вызывающее ввод массива случайными числами

\*/

enum Choices

{

user\_choice = 1,

random\_choice = 2,

};

/\*\*

\* @brief Функция выделяет память под многомерный массив

\* @param rows количество строк в массиве

\* @param cols количество столбцов в массиве

\* @return array указатель на пустой многомерный массив

\*/

int\*\* get\_mem\_array(const size\_t rows, const size\_t cols);

/\*\*

\* @brief Функция заполняет массив, элементы вводит пользователь

\* @param array указатель на заполняемый массив

\* @param rows количество строк в массиве

\* @param cols количество столбцов в массиве

\*/

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

/\*\*

\* @brief Функция заполняет массив рандомными числами в диапазоне, указанный пользователем

\* @param array указатель на заполняемый массив

\* @param rows количество строк в массиве

\* @param cols количество столбцов в массиве

\*/

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

/\*\*

\* @brief Функция, показывающая массив поэлементно

\* @param array ссылка на показываемый массив

\* @param rows количество строк в массиве

\* @param cols количество столбцов в массиве

\*/

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

/\*\*

\* @brief Заменяет отрицателный элемент каждого столбца нулем

\* @param array ссылка на массив

\* @param rows количество строк в массиве

\* @param cols количество столбцов в массиве

\* @return NULL отрицательных чисел нет

\* @return new\_array новый массив, соответствующий этим требованиям

\*/

int \*\*first\_task(int\*\* array, size\_t rows, size\_t cols);

/\*\*

\* @brief Функция удаляет из массива столбцы, содержащие элемент, кратный 5

\* @param array ссылка на массив

\* @param rows количество строк в массиве

\* @param cols количество столбцов в массиве

\* @return result\_array новый массив, соответствующий этим требованиям

\*/

int \*\*second\_task(int\*\* const array, size\_t rows, size\_t cols);

/\*\*

\* @brief Функция вычисляет размер столбцов в массиве, который соответствует условию 2-го пункта

\* @param array ссылка на массив

\* @param rows количество строк в массиве

\* @param cols количество столбцов в массиве

\* @return new\_columns количество столбцов нового массива

\*/

size\_t new\_cols(const int\*\* const array, size\_t rows, size\_t cols);

/\*\*

\* @brief Вспомогательная функция для функции new\_cols, предотвращает выход за диапазон переменной

\* @param number положительное число (или нуль)

\*/

void check\_range\_size\_t(size\_t number);

/\*\*

\* @brief выделяет память под линейный массив

\* @param length длина массива

\* @return array пустой массив

\*/

int\* get\_mem\_array\_linear(const size\_t length);

/\*\*

\* @brief Функция заполняет вспомогательный массив ненужных столбцов

\* @param array многомерный массив

\* @param array\_linear массив, который будет заполняться

\* @param rows количество строк в массиве

\* @param cols количество столбцов в массиве

\*/

void fill\_trash\_columns(const int\*\* const array, int\* array\_linear, size\_t rows, size\_t cols);

/\*\*

\* @brief Функция, проверяющая, что есть ли число в массиве

\* @param array линейный массив

\* @param length длина массива

\* @param number проверяемое число

\* @return true есть число

\* @return false нет числа

\*/

bool contains(const int\* const array, const size\_t length, const size\_t number);

/\*\*

\* @brief точка входа в программу

\* @return 0 программа исправна

\* @return 1 программа завершилась досрочно с ошибкой

\*/

int main()

{

puts("insert a rows\n");

size\_t n = get\_size\_t();

puts("insert a columns\n");

size\_t m = get\_size\_t();

printf("if you fill array by youself, press %d, if you fill array by random numbers, press %d\n", (enum Choices)(user\_choice), (enum Choices)(random\_choice));

int choice = get\_int();

int\*\* mas = get\_mem\_array(n, m);

switch ((enum Choices)choice)

{

case random\_choice:

random\_array(mas, n, m);

break;

case user\_choice:

user\_array(mas, n, m);

break;

default:

puts("Insert a valid choice!\n");

return 1;

}

show\_array(mas, n, m);

int\*\* first\_array = first\_task(mas, n, m);

show\_array(first\_array, n, m);

int\*\* second\_array = second\_task(mas, n, m);

if (NULL == second\_array)

{

puts("Second task == NULL\n");

}

else

{

size\_t new\_columns = new\_cols(mas, n, m);

show\_array(second\_array, n, new\_columns);

}

free\_array(mas, n);

free\_array(first\_array, n);

free\_array(second\_array, n);

return 0;

}

int get\_int()

{

int number;

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

{

puts("Inserted a wrong value\n");

abort();

}

return number;

}

size\_t get\_size\_t()

{

int number = get\_int();

if (number <= 0)

{

errno = ERANGE;

perror("Error: \n");

abort();

}

return (size\_t)number;

}

int\*\* get\_mem\_array(const size\_t rows, const size\_t cols)

{

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

if (NULL == array)

{

array\_null();

}

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

{

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

if (NULL == array[i])

{

array\_null();

}

}

return array;

}

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

{

puts("Insert array elements:\n");

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

{

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

{

array[i][j] = get\_int();

}

}

}

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

{

puts("Enter a minimal number in range\n");

int minimum = get\_int();

puts("Enter a maximum number in range\n");

int maximum = get\_int();

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

{

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

{

array[i][j] = rand() % maximum - minimum;

}

}

}

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

{

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

{

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

{

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

}

printf("\n");

}

puts("\n");

}

int \*\*first\_task(int\*\* array, size\_t rows, size\_t cols)

{

int\*\* new\_array = get\_mem\_array(rows, cols);

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

{

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

{

if (array[i][j] < 0)

{

new\_array[i][j] = 0;

}

else

{

new\_array[i][j] = array[i][j];

}

}

}

return new\_array;

}

void check\_range\_size\_t(size\_t number)

{

if (number == 0)

{

errno = ERANGE;

perror("Error: ");

abort();

}

}

size\_t new\_cols(const int\*\* const array, size\_t rows, size\_t cols)

{

size\_t new\_columns = cols;

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

{

if (array[rows-1][i] > array[-1][i])

{

new\_columns--;

}

}

return new\_columns;

}

int\* get\_mem\_array\_linear(const size\_t length)

{

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

if (NULL == array)

{

array\_null();

}

return array;

}

void fill\_trash\_columns(const int\*\* const array, int\* array\_linear, size\_t rows, size\_t cols)

{

size\_t top = 0;

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

{

if (array[0][i] > array[rows-1][i])

{

array\_linear[top] = i;

top++;

}

}

}

bool contains(const int\* const array, const size\_t length, const size\_t number)

{

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

{

if (number == array[i])

{

return true;

}

}

return false;

}

int \*\*second\_task(int\*\* const array, size\_t rows, size\_t cols)

{

size\_t new\_columns = new\_cols(array, rows, cols);

if (new\_columns == 0)

{

return NULL;

}

int\*\* new\_array = get\_mem\_array(rows, new\_columns);

int\* trash\_columns = get\_mem\_array\_linear(cols - new\_columns);

fill\_trash\_columns(array, trash\_columns, rows, cols);

size\_t row\_top = 0, col\_top = 0;

int flag = 0;

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

{

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

{

if (!contains(trash\_columns, (cols - new\_columns), i))

{

new\_array[i][col\_top] = array[j][i];

col\_top++;

}

}

}

if (NULL != trash\_columns)

{

free(trash\_columns);

}

return new\_array;

}

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

{

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

{

if (NULL != array[i])

{

free(array[i]);

}

}

if (NULL != array)

{

free(array);

}

}

void array\_null()

{

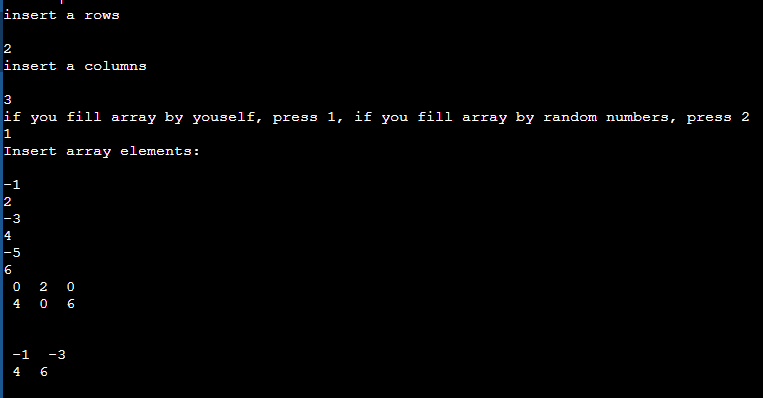
errno = ENOMEM;

perror("Error: ");

abort();

}

3.4 Результаты выполнения программы

  
Рисунок 57 - Результаты выполнения программы

3.5 Выполнение тестовых примеров

3.6 Отметка о выполнении задания в веб-хостинге системы контроля версий

Рисунок 58 – Результат выполнения задания в веб хостинге