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 №5

По дисциплине «Языки программирования»

ВАРИАНТ 17

Выполнил: ст. гр. ТКИ-141

Лагутин Владимир Сергеевич

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

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

Москва 2024

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 – Исходные данные

|  |  |  |
| --- | --- | --- |
| Вариант | Задачи | Интервал |
| 17 | 1. Найти сумму элементов, имеющих нечетное значение. 2. Вывести индексы тех элементов, значения которых больше заданного числа А. 3. Заменить второй элемент массива на максимальный среди отрицательных. | [-10; 10] |

1. Блок-схема алгоритма

Блок-схема основного алгоритма представлена ниже (Рисунок 1). Блок-схемы функций расчета значений a и b представлены ниже (Рисунок 2).

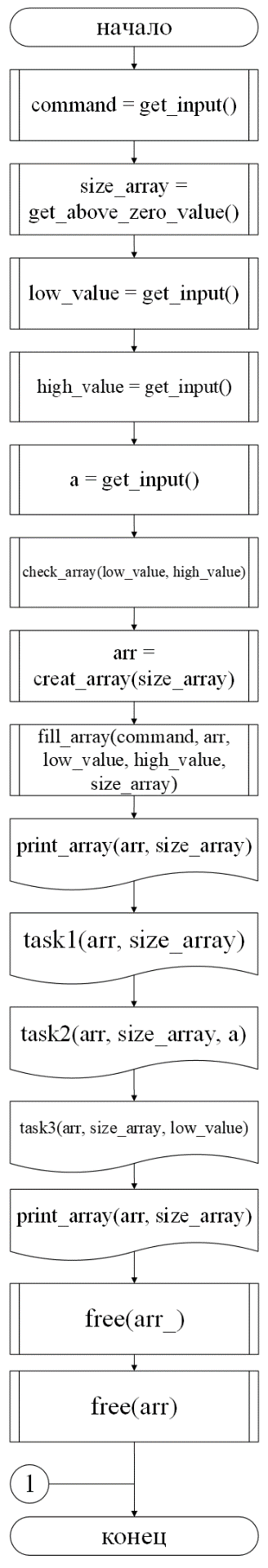
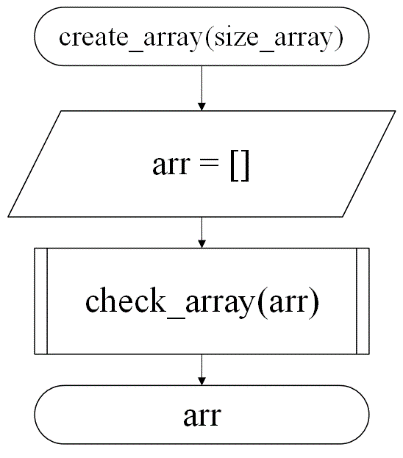
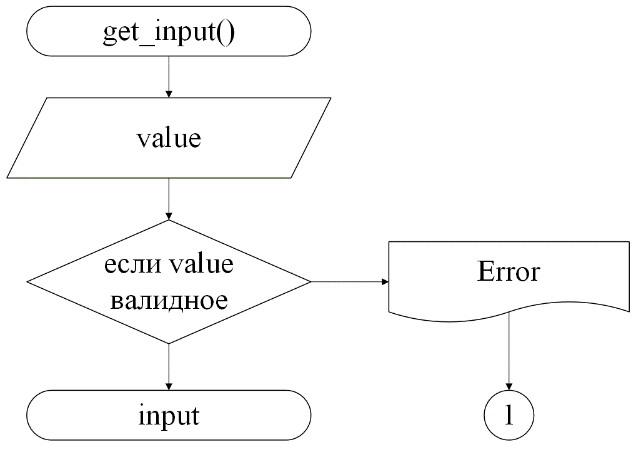
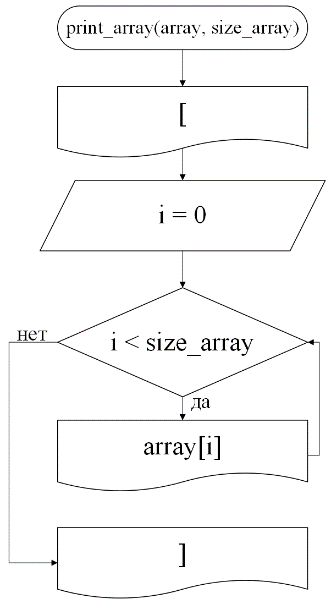
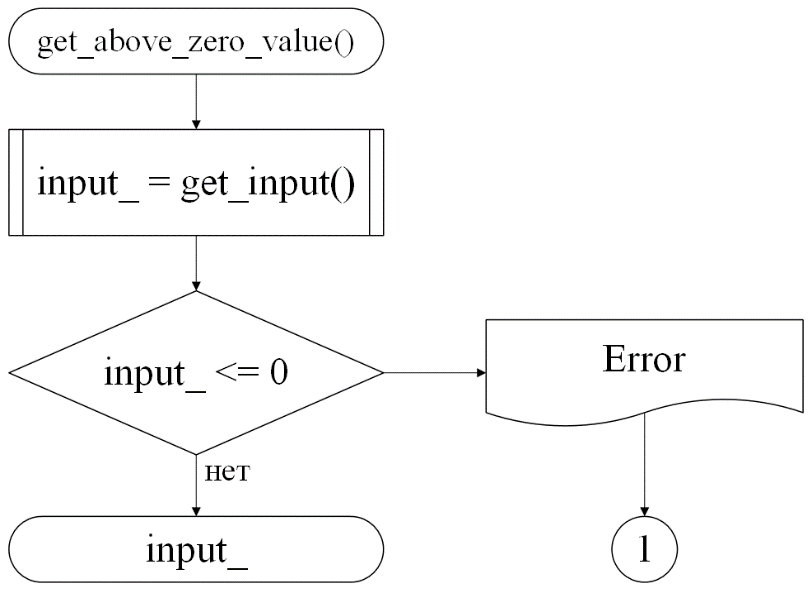
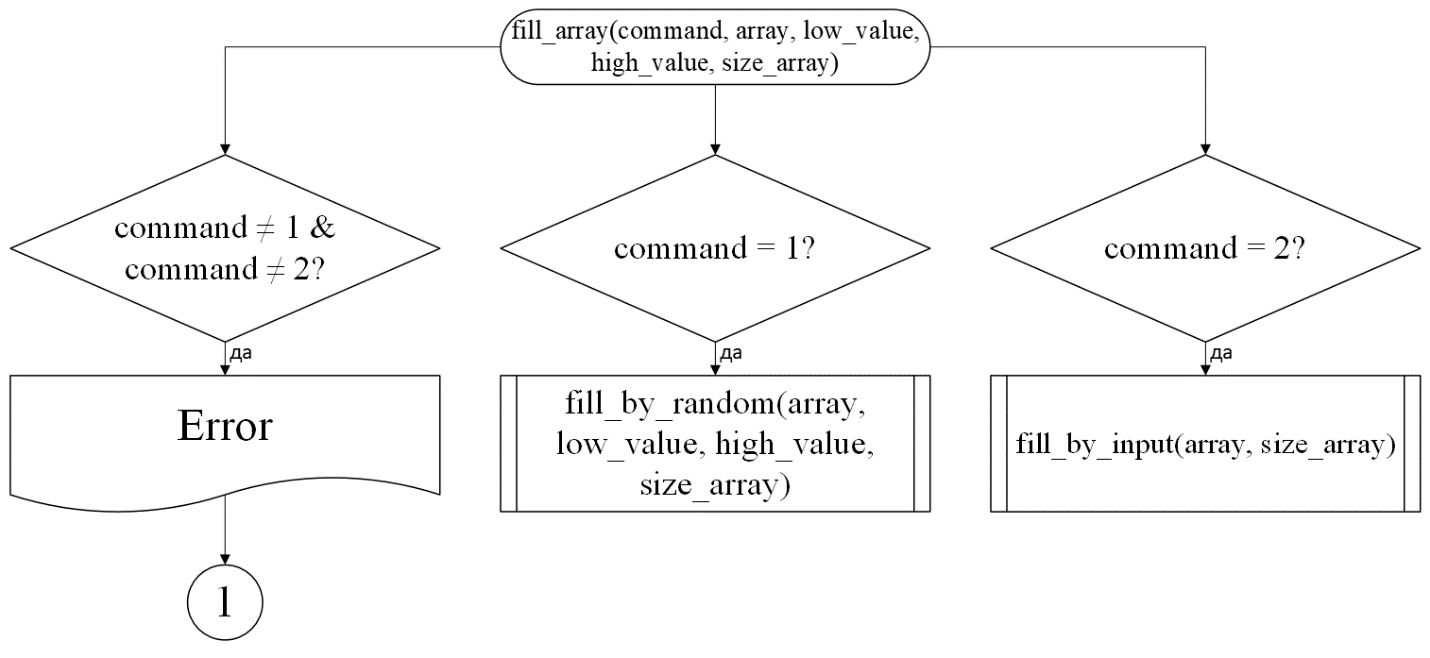
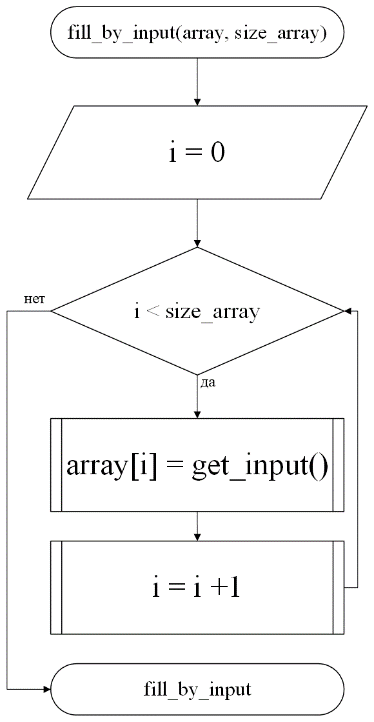
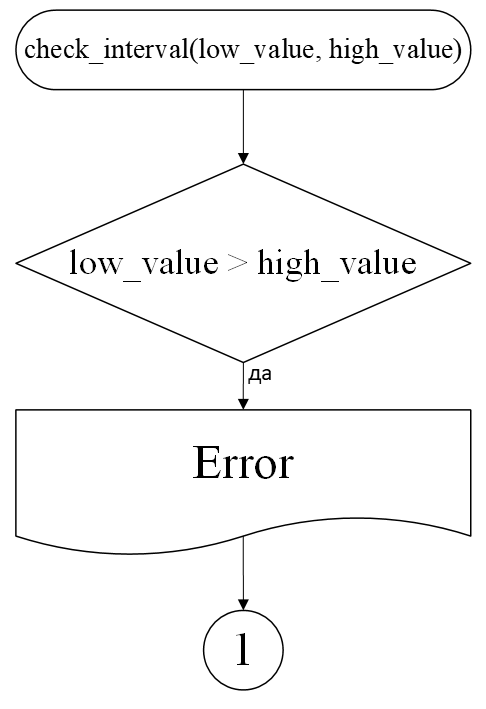
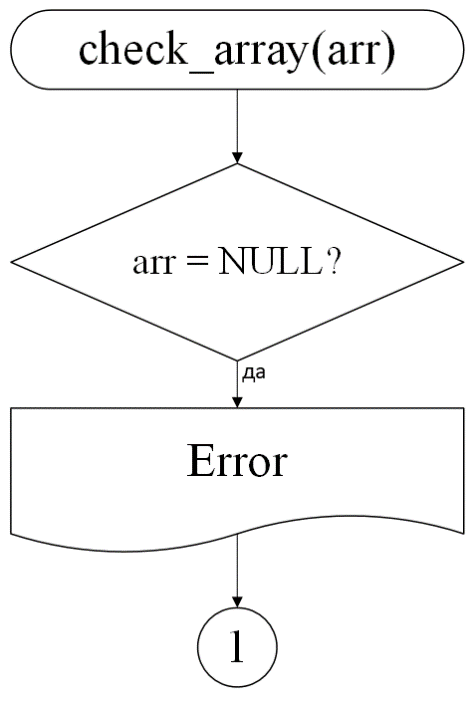
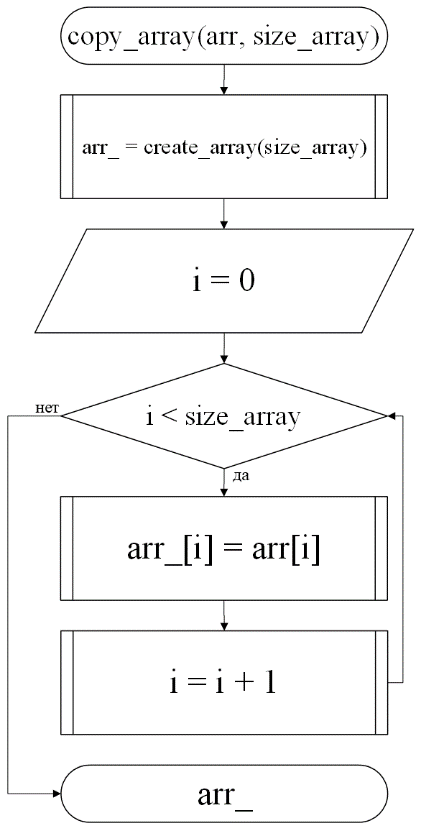
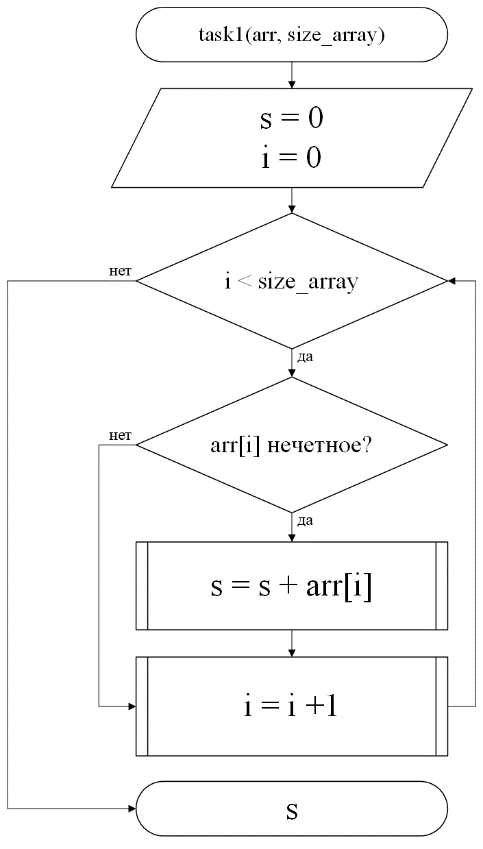
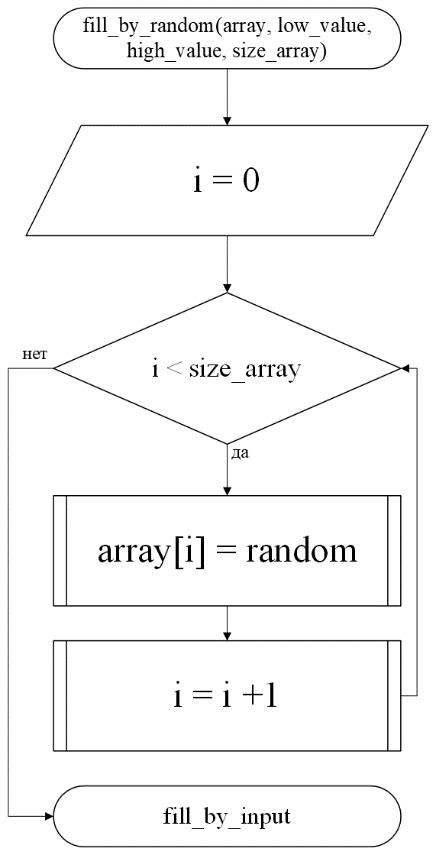
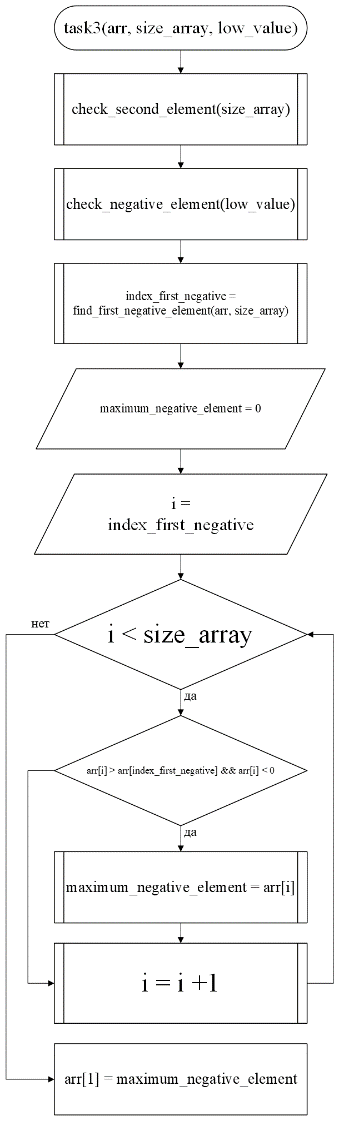
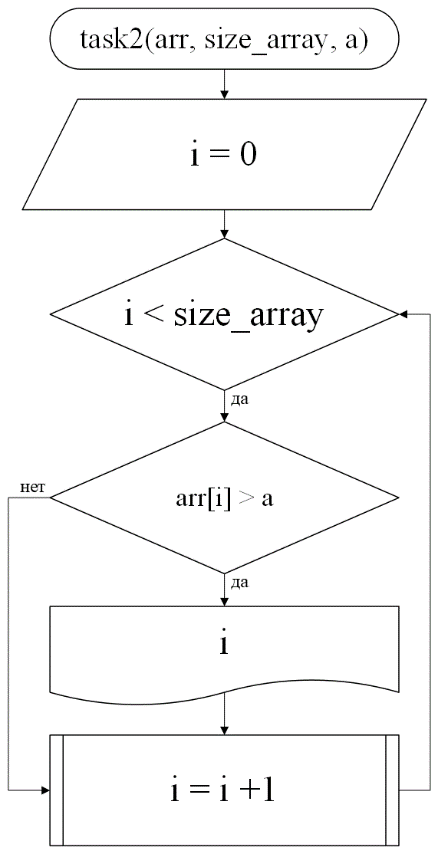


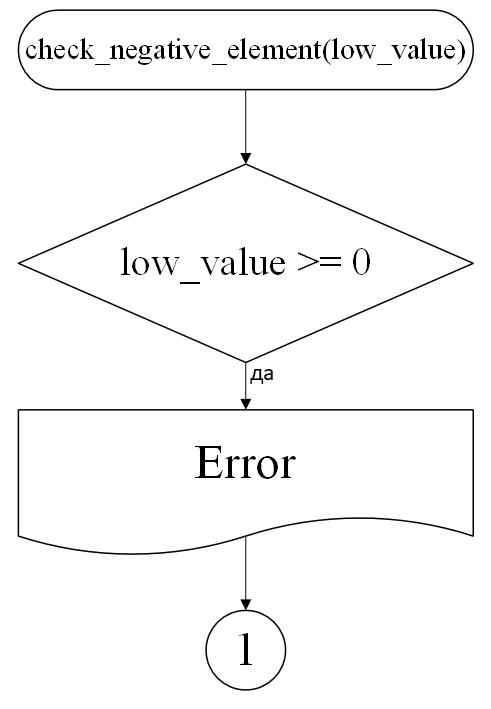
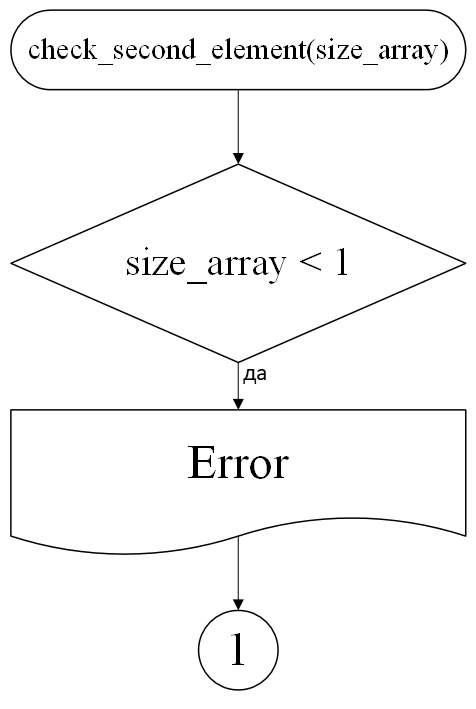
Рисунок 1 ­ Блок-схема основного алгоритма











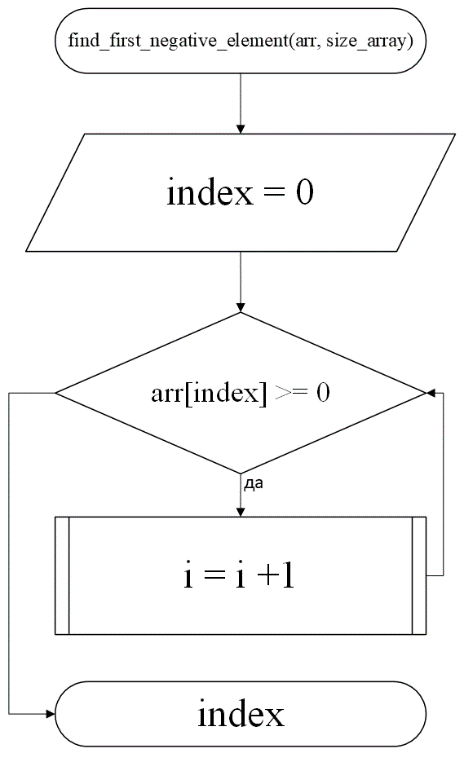


Рисунок 2 – Блок-схема используемых функций

1. Текст программы на языке C

#include <stdio.h>

#include <stdlib.h>

#include <math.h>

#include <time.h>

#include <locale.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 get\_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, int const a);

/\*\*

\* **@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, const int low\_value);

/\*\*

\* **@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 2-го элемента для задания 3

\* **@remarks** Завершает программу с кодом ошибки /c EXIT\_FAILURE если в массиве меньше 2-х элементов

\*/

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

/\*\*

\* **@brief** Функция проверки наличия отрицательного элемента для задания 3

\* **@remarks** Завершает программу с кодом ошибки /c EXIT\_FAILURE если в массиве нет отрицательного элемента

\*/

void check\_negative\_element(const int low\_value);

/\*\*

\* **@brief** Функция нахождения первого отрицательного элемента для задания 3

\* **@return** Индекс первого отрицательного элемента

\*/

int find\_first\_negative\_element(int\* const arr, 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 = 2,

};

/\*\*

\* **@brief** Точка входа

\* **@return** Возвращает код ошибки 0

\*/

int main(void) {

setlocale(LC\_ALL, "Russian");

srand(time(NULL));

printf\_s("Как Вы хотите заполнить массив ? \n%d - слуйчайный ввод, %d - ручной ввод\n", fill\_random, fill\_input);

int command = get\_input();

puts("Введите размер массива");

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

puts("Введите меньшее и большее значение массива (через пробел):");

int low\_value = get\_input(), high\_value = get\_input();

puts("Введите число А:");

int a = get\_input();

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

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

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

puts("Массив:");

print\_array(arr, size\_array);

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

printf("Задание 1: %d\n", task1(arr\_, size\_array));

puts("Задание 2: ");

task2(arr\_, size\_array, a);

puts("Задание 3: ");

task3(arr\_, size\_array, low\_value);

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("Входной сигнал памяти");

exit(EXIT\_FAILURE);

}

}

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

if (low\_value > high\_value) {

puts("Ошибка интервала");

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("Ошибка команды");

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] = get\_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 get\_input(void) {

int value = 0;

int result = scanf\_s("%d", &value);

if (result != 1) {

puts("Неправильное введеное значение!");

exit(EXIT\_FAILURE);

}

return value;

}

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

int input\_ = get\_input();

if (input\_ <= 0) {

puts("Неправильное введеное значение!");

exit(EXIT\_FAILURE);

}

return (size\_t)input\_;

}

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

printf("[");

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 (arr[i] % 2 != 0) {

s += arr[i];

}

}

return s;

}

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

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

if (arr[i] > a) {

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

}

}

}

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

check\_second\_element(size\_array);

check\_negative\_element(low\_value);

int index\_first\_negative = find\_first\_negative\_element(arr, size\_array);

int maximum\_negative\_element = 0;

for (size\_t i = index\_first\_negative + 1; i < size\_array; ++i) {

if (arr[i] > arr[index\_first\_negative] && arr[i] < 0) {

maximum\_negative\_element = arr[i];

}

}

arr[1] = maximum\_negative\_element;

}

void check\_second\_element(const size\_t size\_array) {

if (size\_array < 1) {

printf("Задание 3 не может быть выполнено!");

exit(EXIT\_FAILURE);

}

}

void check\_negative\_element(const int low\_value) {

if (low\_value >= 0) {

printf("Задание 3 не может быть выполнено!");

exit(EXIT\_FAILURE);

}

}

int find\_first\_negative\_element(int\* const arr, const size\_t size\_array) {

int index = 0;

while (arr[index] >= 0) {

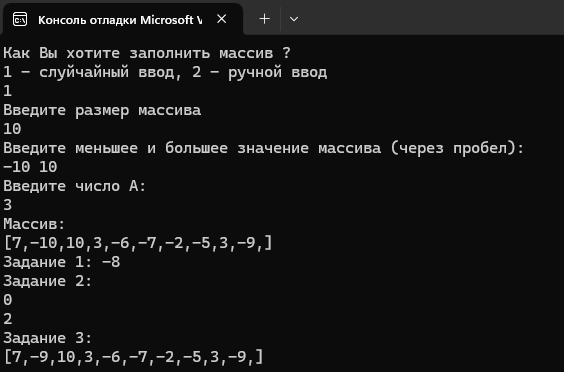
index++;

}

return index;

}

1. Результаты выполнения программы

Результаты выполнения программы представлены ниже (Рисунок 3).

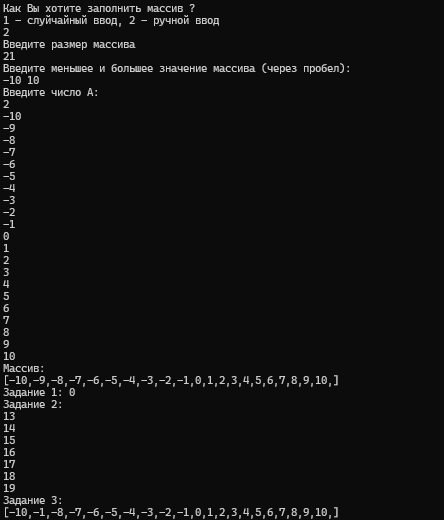
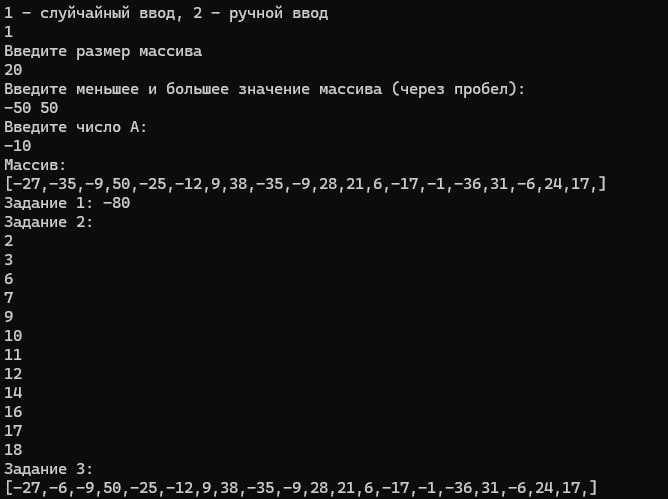


Рисунок 3 – Результаты выполнения программы

1. Выполнение тестовых примеров

Выполнены тестовые примеры. Результаты их выполнения представлены ниже (Рисунок 4, Рисунок 5). Выполнены тестовые примеры с неправильными входными данными (Рисунок 7).Рисунок  – Результат тестового расчета

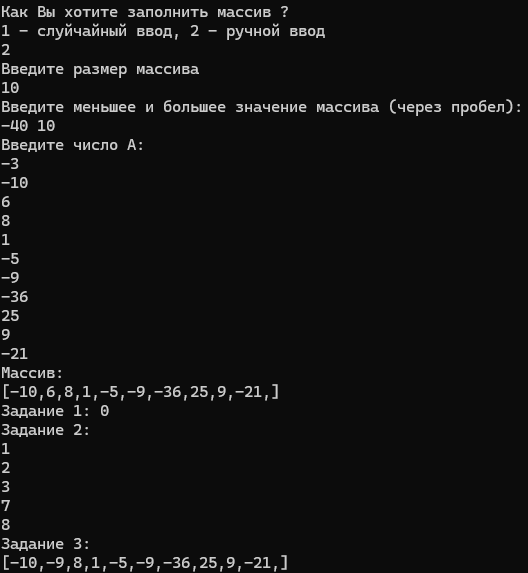
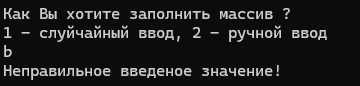
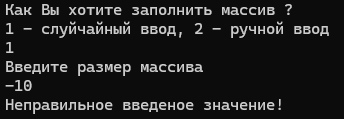
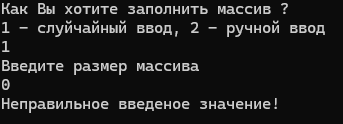
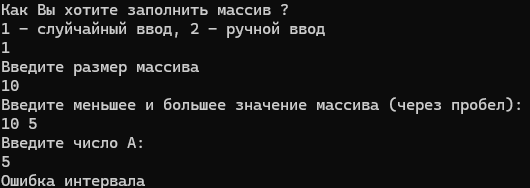


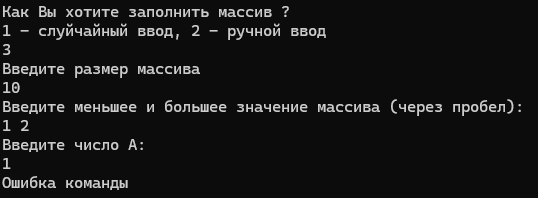
Рисунок 5 – Результат тестового расчета

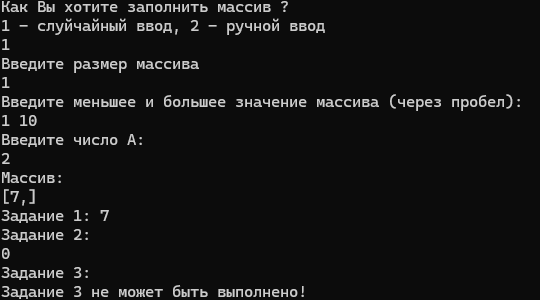












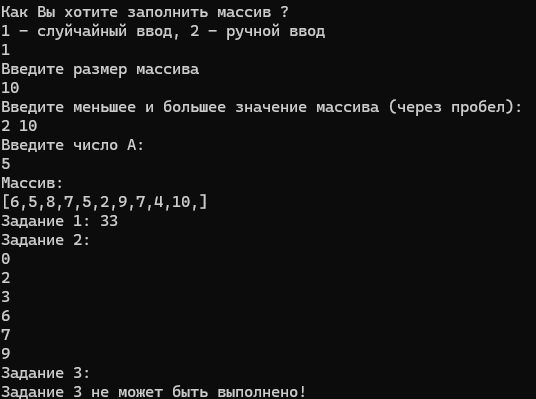


Рисунок 6 – Выполнение тестовых примеров с неправильными входными данными

1. Отметка о выполнении задания в веб-хостинге системы контроля версий

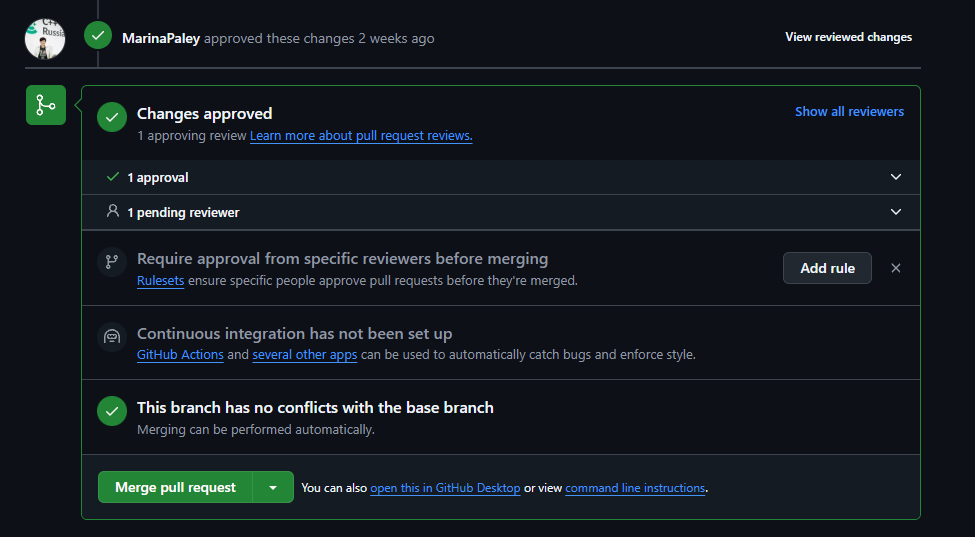


Рисунок 7 – Одобренный код по заданию 3-1