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 5

Выполнил: ст. гр. ТКИ-142

Гаспарян Артём Артурович

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

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

Москва 2024

**Оглавление**

[1 ЗАДАНИЕ 4.1 3](#_Toc185169371)

[1.1 Формулировка задания 3](#_Toc185169372)

[1.2 Блок-схема основного алгоритма 4](#_Toc185169373)

[1.3 Текст программы на языке С 17](#_Toc185169374)

[1.4 Результат выполнения программы 22](#_Toc185169375)

[1.5 Отметка о выполнение программы 27](#_Toc185169376)

[2 2 ЗАДАНИЕ 4.2 28](#_Toc185169377)

[2.1 Формулировка задания 28](#_Toc185169378)

[2.2 Блок-схема основного алгоритма 29](#_Toc185169379)

1. ЗАДАНИЕ 4.1
   1. Формулировка задания

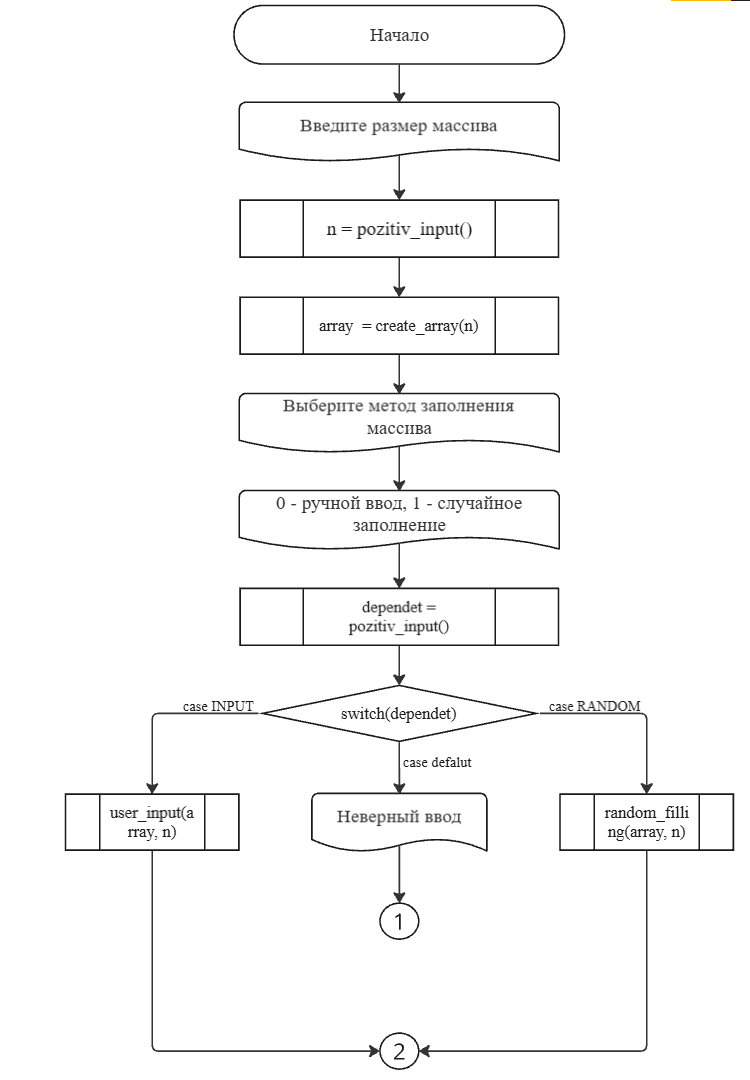
Создать одномерный массив из *n* целых чисел. Реализовать возможность заполнения массива, как случайными числами, так и с помощью клавиатуры по желанию пользователя. Пункты задания организовать в виде функций (методов). Составить блок-схему (Таблица 1).

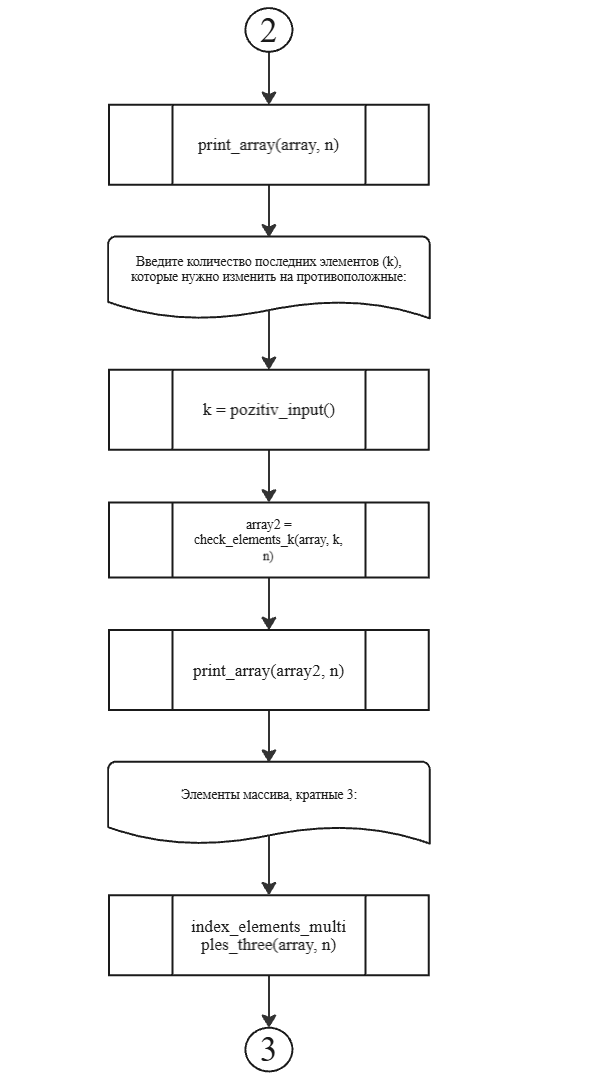
Таблица 1 – Исходные данные

|  |  |  |
| --- | --- | --- |
| **Вариант** | **Задание** | **Интервал** |
| **5** | 1. Заменить последние k элементов массива на противоположные по знаку.  2. Вывести индексы тех элементов, значения которых кратны 3.  3. Определить, есть ли пара соседних элементов с суммой, равной заданному числу. | [-100;100] |

* 1. Блок-схема основного алгоритма

Ниже представлена блок-схема основного алгоритма (Рисунок 1). Блок-схемы функция расчета (Рисунок 2, Рисунок 3, Рисунок 4, Рисунок 5, Рисунок 6, Рисунок 7, Рисунок 8, Рисунок 9, Рисунок 10, Рисунок 11, Рисунок 12).





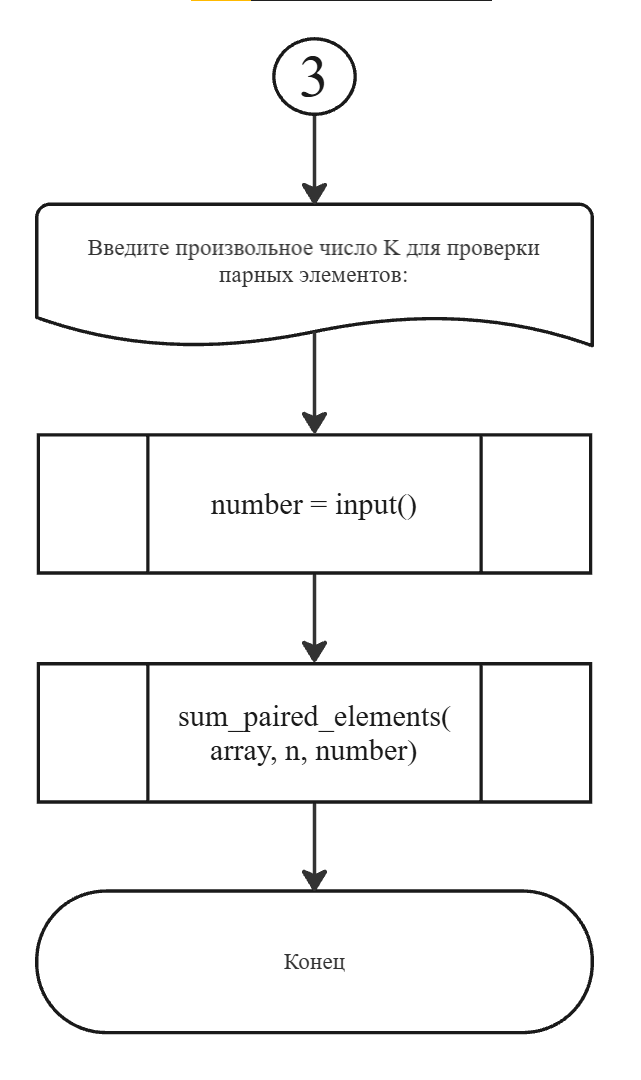


Рисунок 1 – Блок-схема основного алгоритма

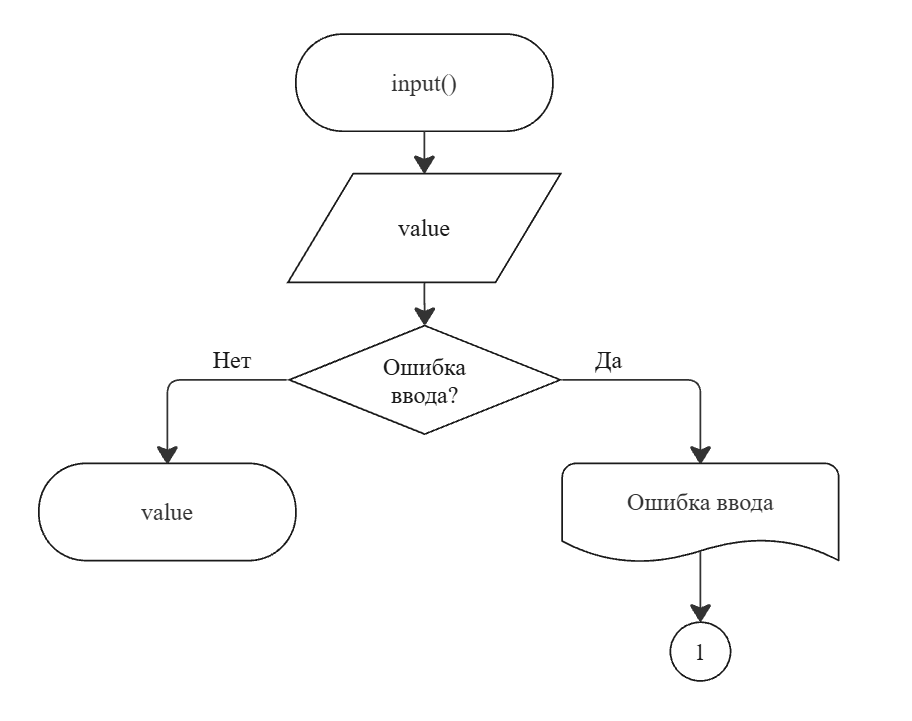


Рисунок 2 – Блок-схема функции input()

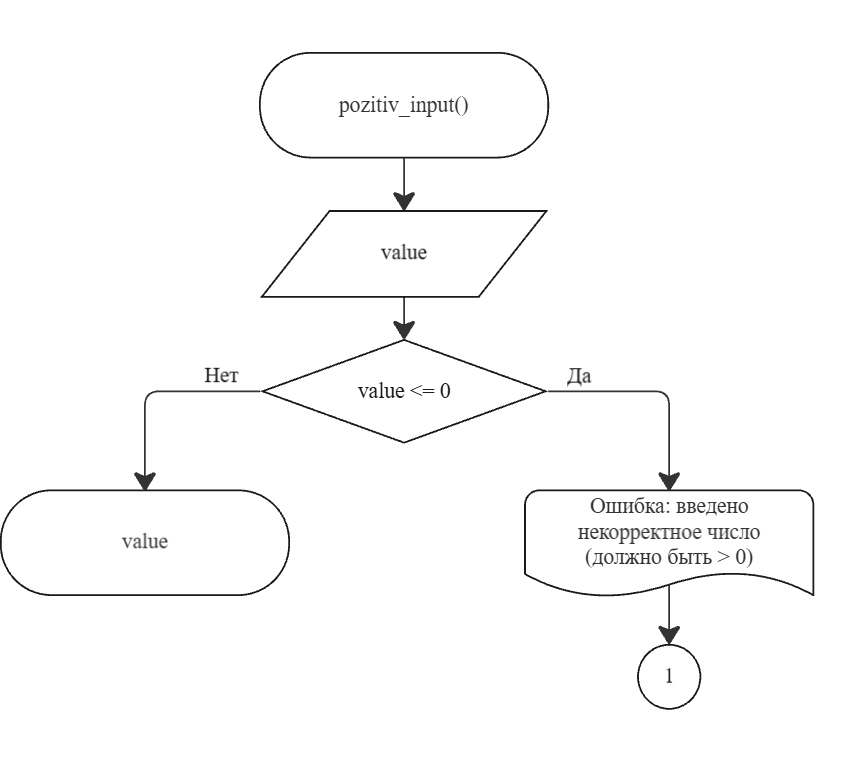


Рисунок 3 – Блок-схема функции pozitiv\_input()

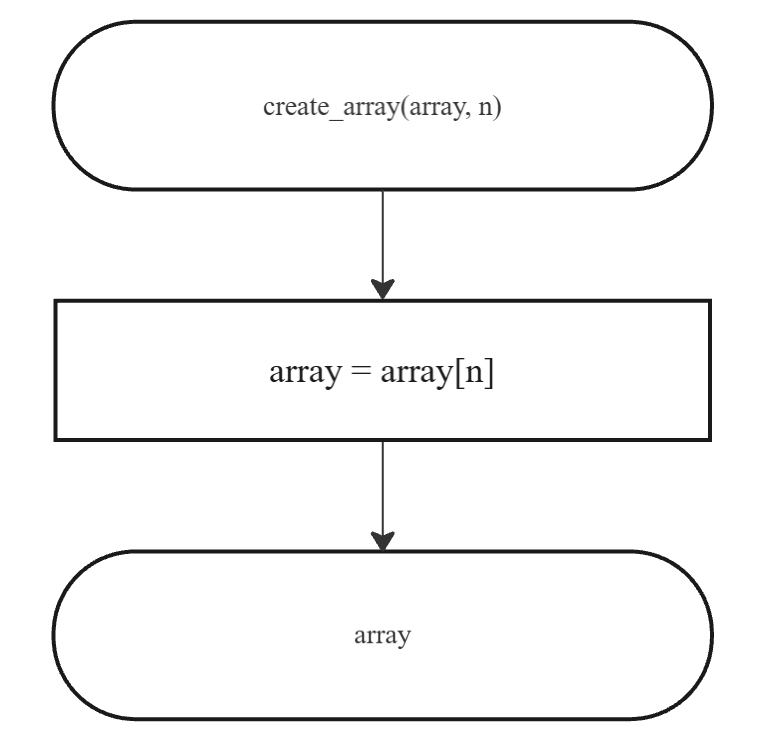


Рисунок 4 – Блок-схема функции create\_array(array, n)

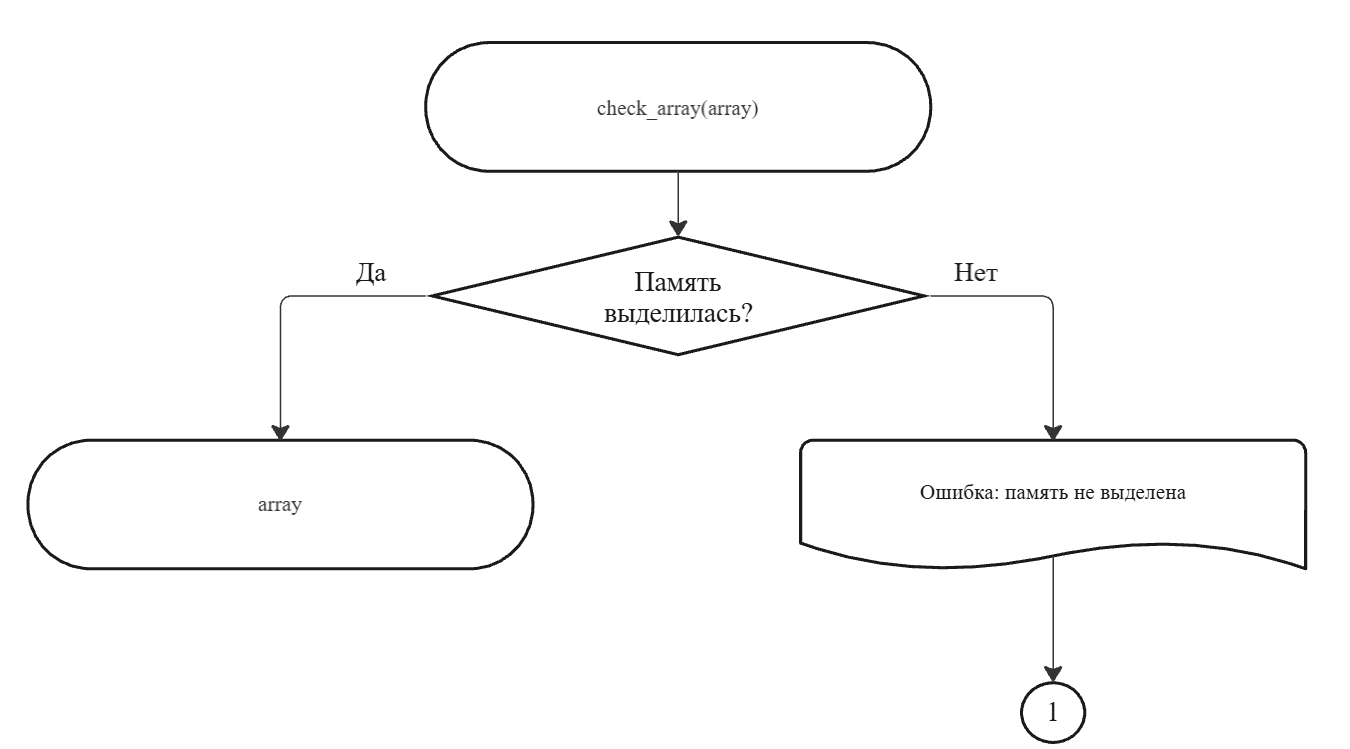


Рисунок 5 – Блок-схема функции check\_array(array)

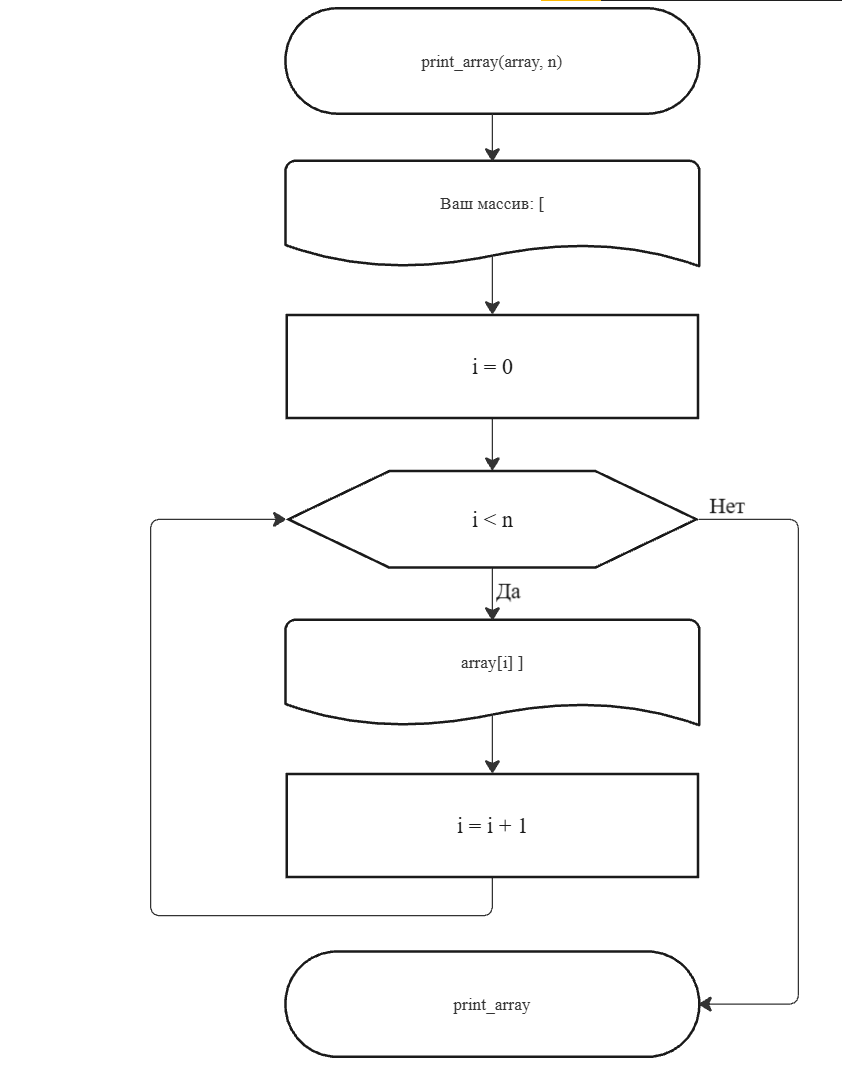


Рисунок 6 – Блок-схема функции print\_array(array, n)

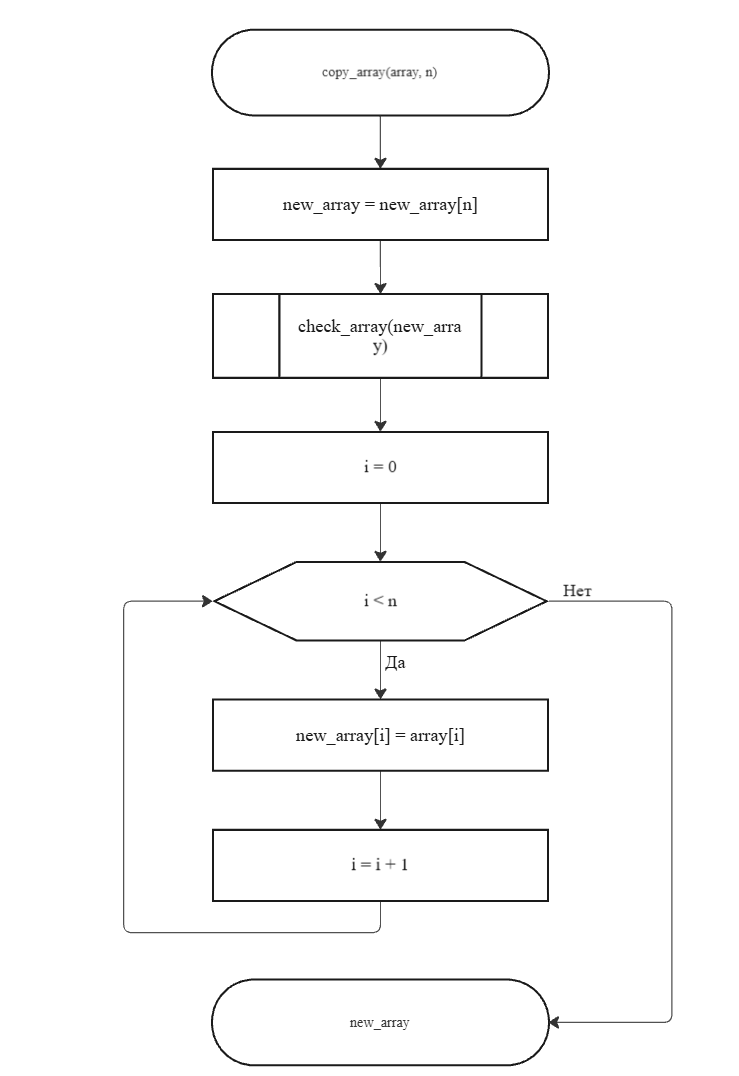


Рисунок 7 – Блок-схема функции copy\_array(array, n)

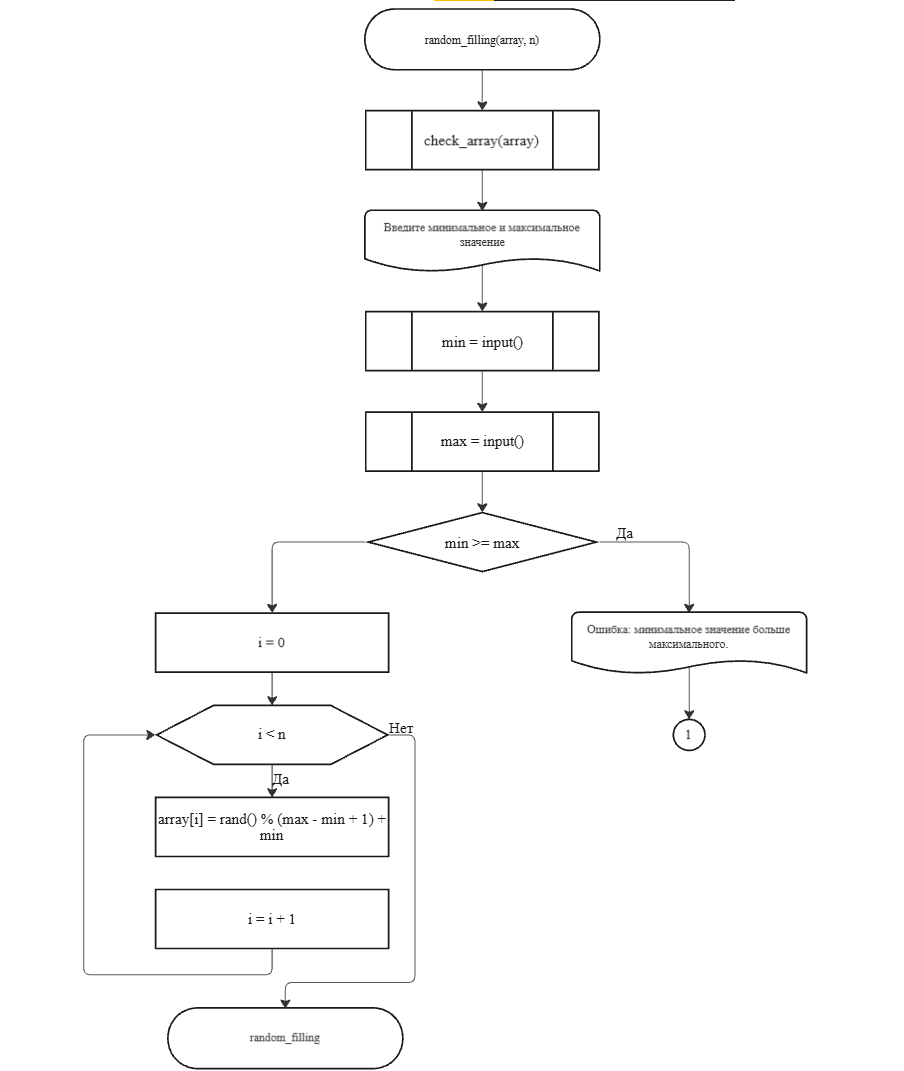


Рисунок 8 – Блок-схема функции random\_filling(array, n)

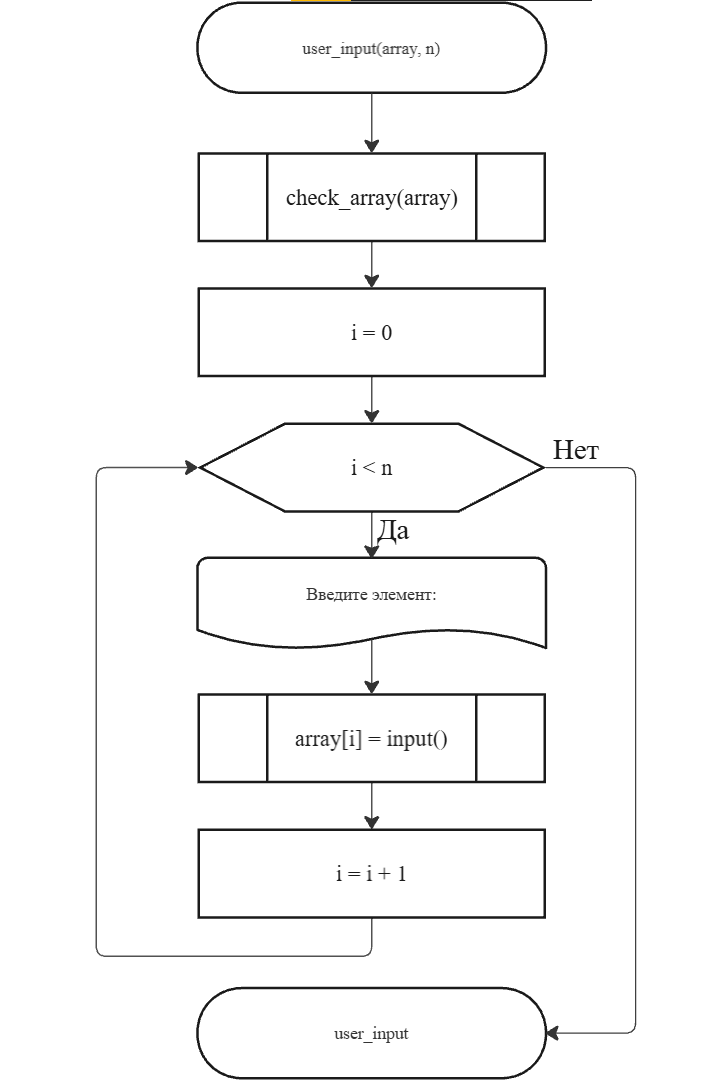


Рисунок 9 – Блок-схема функции user\_input(array, n)

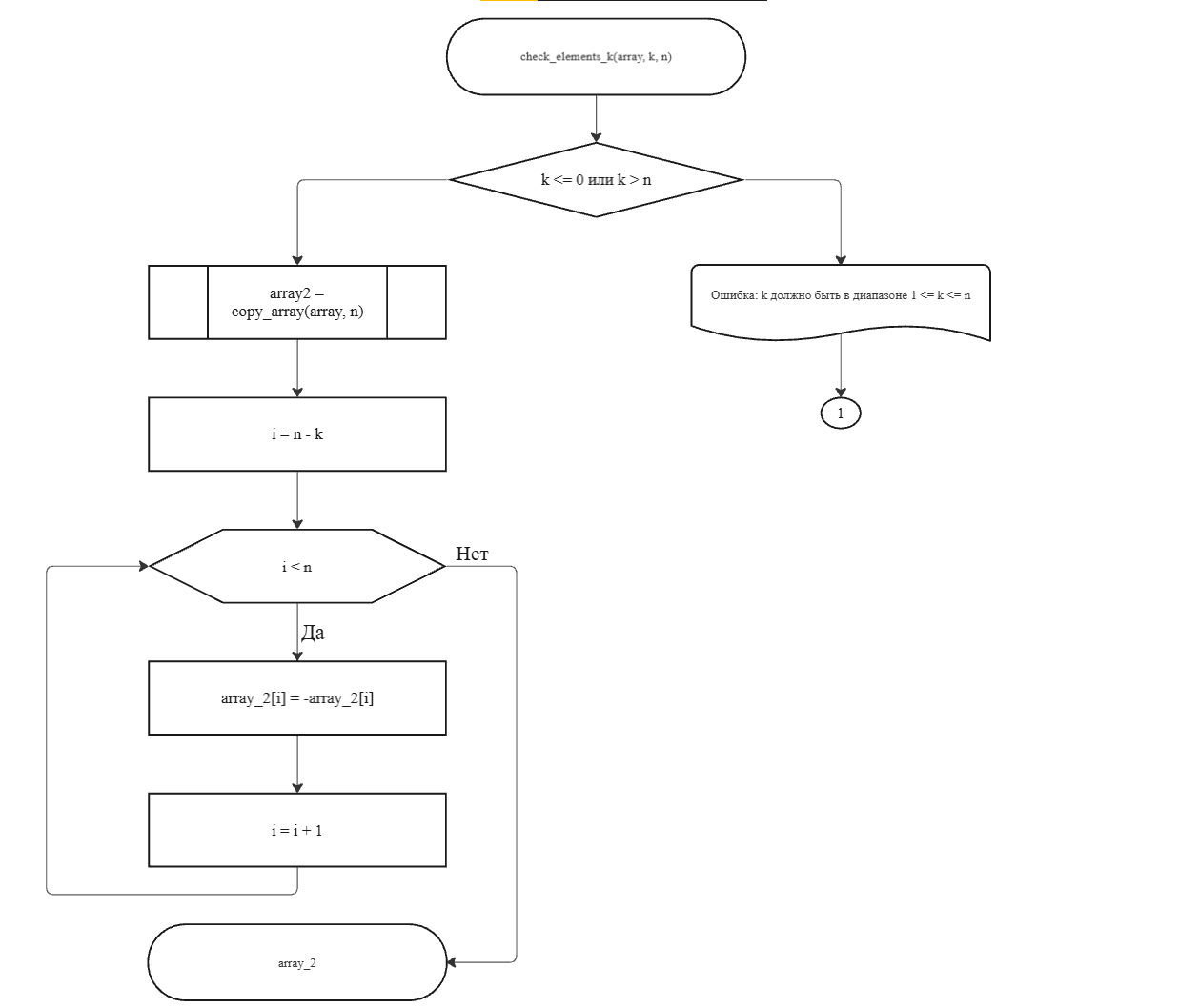


Рисунок 10 – Блок-схема функции check\_elements\_k(array, k, n)

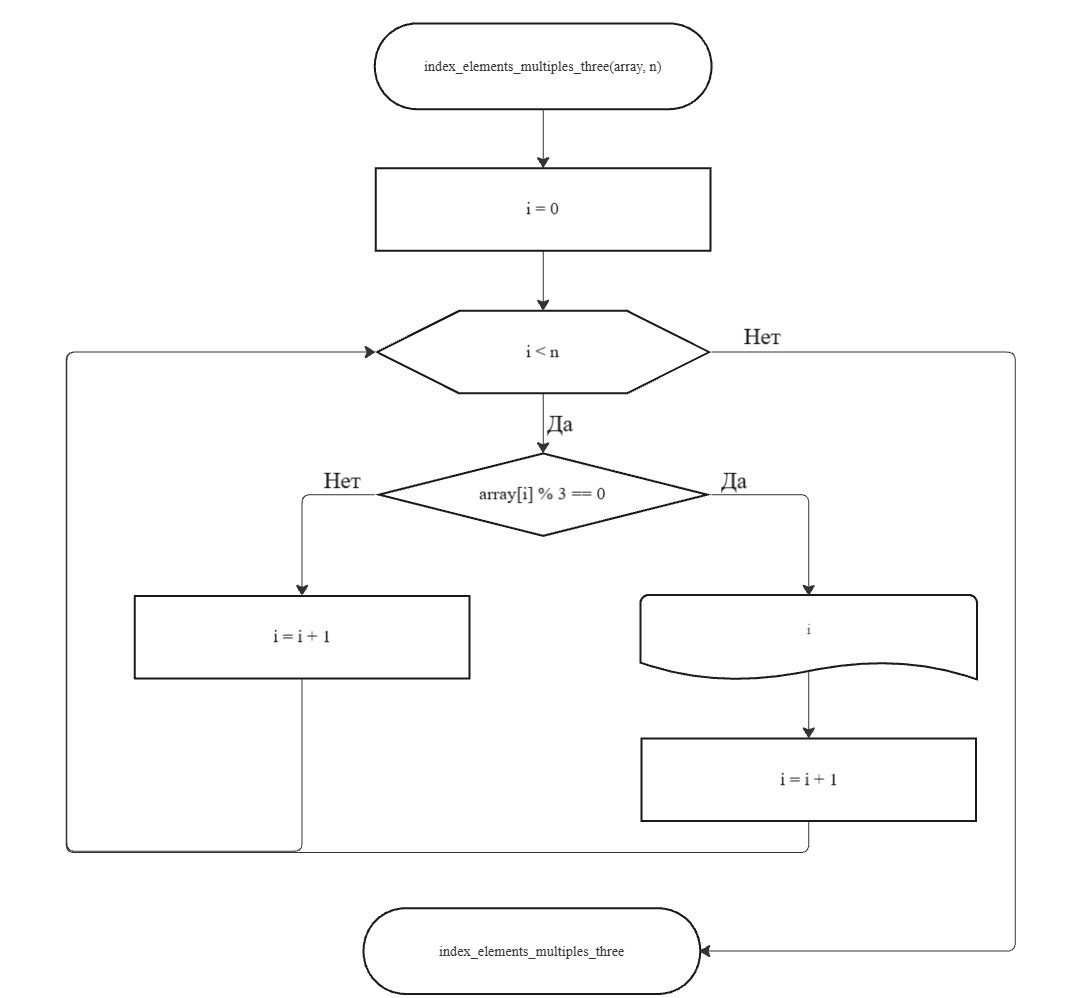


Рисунок 11 – Блок-схема функции index\_elements\_multiples\_three(array, n)

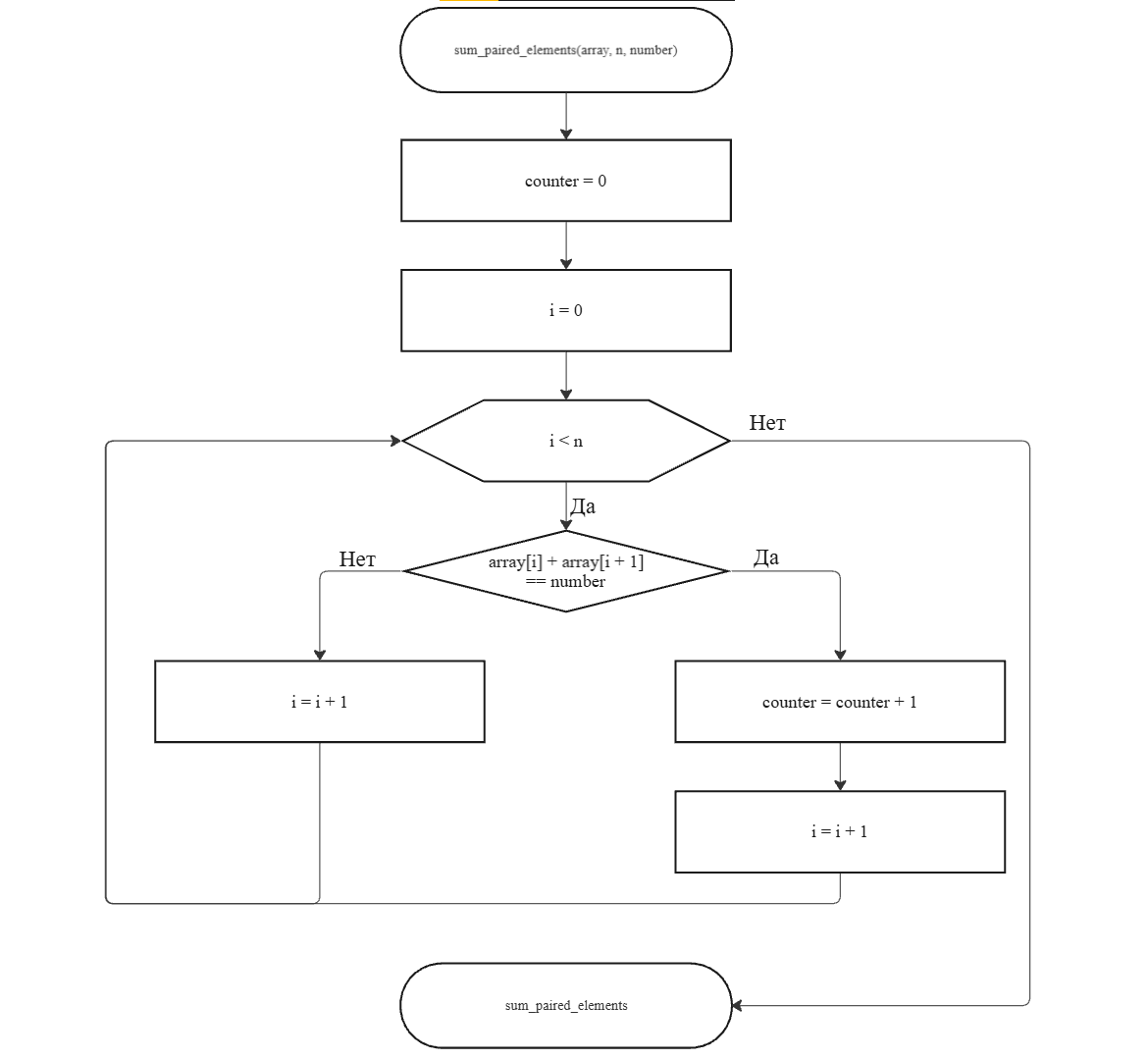


Рисунок 12 – Блок-схема функции sum\_paired\_elements(array, n, number)

1.3 Текст программы на языке С

#include <stdio.h>

#include <stdlib.h>

#include <math.h>

#include <errno.h>

#include <time.h>

#include <locale.h>

/\*\*

\* @brief функция, выполняющая проверку введенного пользователем числа

\* @return ошибку в случае, если пользователь ввел не число или ввел не одно значение

\*/

int input(void);

/\*\*

\* @brief функция, выполняющая проверку на положительное число

\* @return ошибку в случае, если пользователь ввел отрицательное число

\*/

int pozitiv\_input(void);

/\*\*

\* @brief функция, создающая массив

\* @param n переменная, отвечающая за размер массива

\* @return возвращает созданный массив

\*/

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

/\*\*

\* @param INPUT отвечате за заполнение массива из потока ввода

\* @param RANDOM отвечает за рандомнное заполнение массива

\*/

enum choice{

INPUT = 0, RANDOM = 1

};

/\*\*

\* @brief заполняет массив рандомными числами

\* @param array - указатель на массив (эквивалентно array[])

\* @param n переменная, отвечающая за размер массива

\* @param min переменная, хранящая в себе минимальное число

\* @param max переменная, хранящая в себе максимальное число

\*/

void random\_filling(int \*array, const size\_t n);

/\*\*

\* @brief заполняет массив пользовательским вводом

\* @param array - указатель на массив (эквивалентно array[])

\* @param n перемнная, отвечающая за размер массива

\*/

void user\_input(int \*array, const size\_t n);

/\*\*

\* @brief функция которая печатает массив

\* @param array указатель на массив (эквивалентно array [])

\* @param n перменная, отвечающая за размер массива

\*/

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

/\*\*

\* @brief функция, проверяющая адресс

\* @return в случае, если адрессс нулевой, программа закрывается

\*/

void check\_array(int \* array);

/\*\*

\* @brief создает новый массив на копии старого

\* @param array указатель на массив

\*/

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

/\*\*

\* @brief заменяет последние k элементов массива на противоположные

\* @param array - указатель на массив (эквиваленто array[])

\* @param k переменная k

\* @param n переменная, отвечающая за размер массива

\*/

int\* check\_elements\_k(int \*array, const int k, const size\_t n);

/\*\*

\* @brief функция находит элементы, кратные 3ом, и выводит их

\* @param array указатель на массив (эквивалентно array[])

\* @param n переменная, отвечающая за размер массива

\*/

void index\_elements\_multiples\_three(const int \*array, const size\_t n);

/\*\*

\* @brief функция определяет есть ли пара соседних элементов с суммой, равной заданной заданному число

\* @param array указатель на массив (эквивалентно array[])

\* @param n переменная, отвечающая за размер массива

\* @param k переменная, которая хранит в себе произвольное число

\*/

void sum\_paired\_elements(int \*array, const size\_t n, const int k);

#include "4.h"

int main(void) {

printf("Введите размер массива\n");

const size\_t n = (size\_t)pozitiv\_input();

int\* array = create\_array(n);

printf("Выберите метод заполнения массива.\n");

printf("0 - ручной ввод, 1 - случайное заполнение\n");

int dependet = pozitiv\_input();

switch (dependet) {

case INPUT:

user\_input(array, n);

break;

case RANDOM:

random\_filling(array, n);

break;

default:

printf("Неверный ввод.\n");

free(array);

exit(EXIT\_FAILURE);

}

print\_array(array, n);

printf("Введите количество последних элементов (k), которые нужно изменить на противоположные:\n");

const int k = pozitiv\_input();

int\* array2 = check\_elements\_k(array, k, n);

print\_array(array2, n);

free(array2);

printf("Элементы массива, кратные 3:\n");

index\_elements\_multiples\_three(array, n);

printf("Введите произвольное число K для проверки парных элементов:\n");

const int number = input();

sum\_paired\_elements(array, n, number);

free(array);

return 0;

}

int input(void) {

int value;

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

perror("Ошибка ввода");

exit(EXIT\_FAILURE);

}

return value;

}

int pozitiv\_input(void) {

int value = input();

if (value <= 0) {

perror("Ошибка: введено некорректное число (должно быть > 0)");

exit(EXIT\_FAILURE);

}

return value;

}

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

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

check\_array(array);

return array;

}

void random\_filling(int \*array, const size\_t n) {

check\_array(array);

printf("Введите минимальное и максимальное значение:\n");

int min = input();

int max = input();

if (min > max) {

printf("Ошибка: минимальное значение больше максимального.\n");

exit(EXIT\_FAILURE);

}

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

array[i] = rand() % (max - min + 1) + min;

}

}

void user\_input(int \*array, const size\_t n) {

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

printf("Введите %zu-й элемент:\n", i + 1);

array[i] = input();

}

}

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

printf("Ваш массив: [");

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

printf("%d%s", array[i], (i < n - 1) ? ", " : "]\n");

}

}

void check\_array(const int \*array) {

if (!array) {

perror("Ошибка: массив не выделен");

exit(EXIT\_FAILURE);

}

}

int\* copy\_array(const int \*array, const size\_t n) {

int\* new\_array = (int\*)malloc(n \* sizeof(int));

check\_array(new\_array);

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

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

}

return new\_array;

}

int\* check\_elements\_k(const int \*array, int k, const size\_t n) {

if (k <= 0 || k > (int)n) {

printf("Ошибка: k должно быть в диапазоне 1 <= k <= %zu\n", n);

exit(EXIT\_FAILURE);

}

int\* array\_2 = copy\_array(array, n);

for (size\_t i = n - k; i < n; ++i) {

array\_2[i] = -array\_2[i];

}

return array\_2;

}

void index\_elements\_multiples\_three(const int \*array, const size\_t n) {

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

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

printf("Индекс: %zu, значение: %d\n", i, array[i]);

}

}

}

void sum\_paired\_elements(const int \*array, const size\_t n, const int number) {

size\_t count = 0;

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

if (array[i] + array[i + 1] == number) {

count++;

}

}

printf("Найдено %zu пар, сумма которых равна %d\n", count, number);

}

1.4 Результат выполнения программы

Результаты выполнения программы представлены ниже (Рисунок 13, Рисунок 14, Рисунок 15), также тестовые примеры, которые могут потенциально сломать программу (Рисунок 16, Рисунок 17, Рисунок 18, Рисунок 19, Рисунок 20, Рисунок 21, Рисунок 22, Рисунок 23, Рисунок 24, Рисунок 25, Рисунок 26, Рисунок 27, Рисунок 28, Рисунок 29).

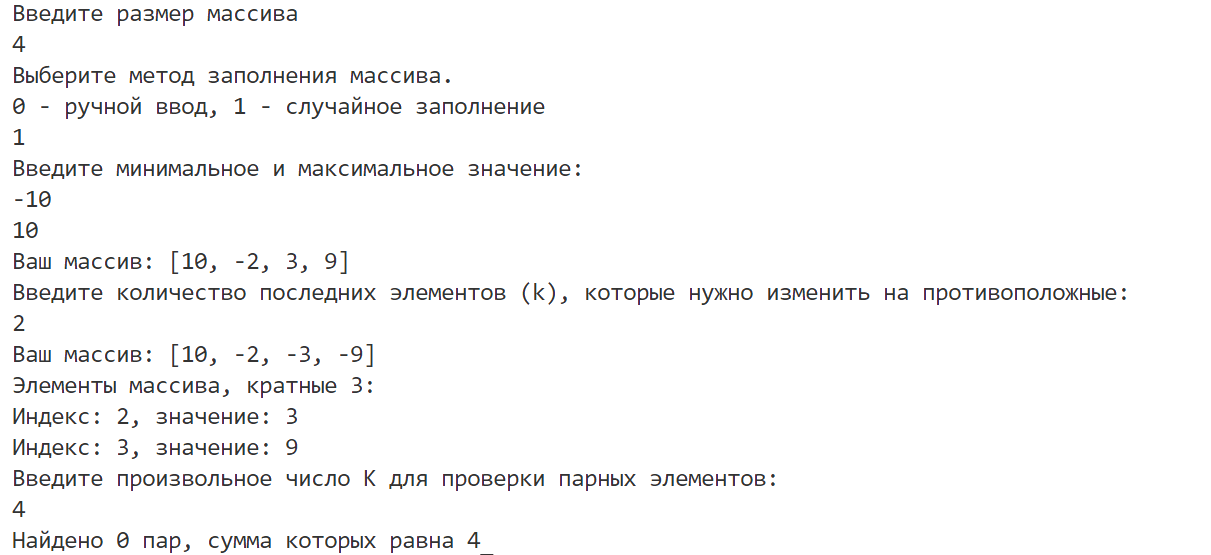


Рисунок 13 – Первый результат выполнения программы

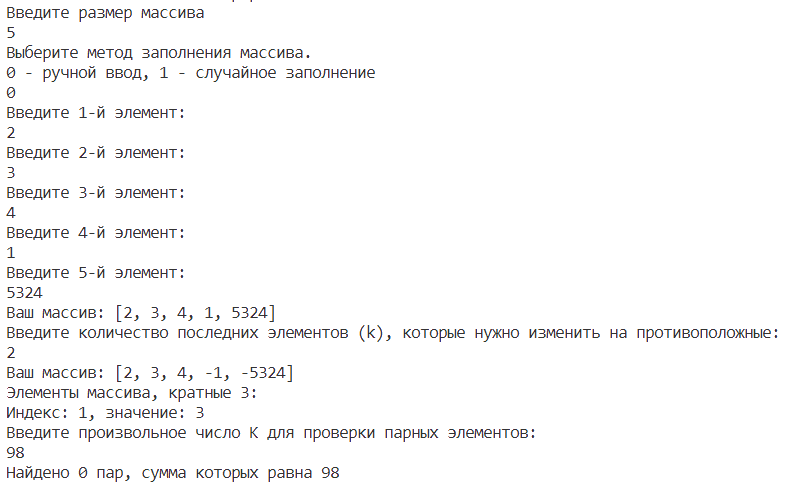


Рисунок 14 – Второй результат выполнения программы

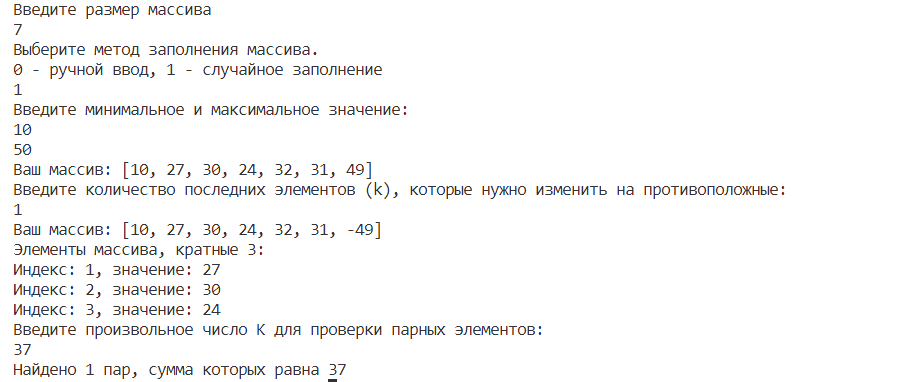


Рисунок 15 ‬– Третий результат выполнения программы

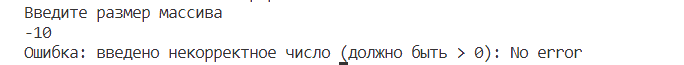


Рисунок 16 – Проверка функции pozitiv\_input()

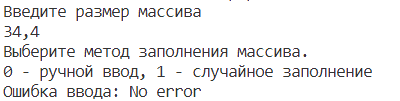


Рисунок 17 – Вместо int вводим double

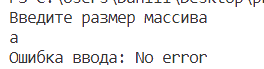


Рисунок 18 – Вместо int вводим буквы (char)

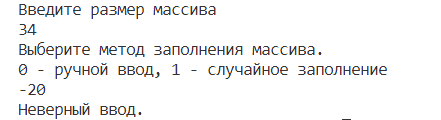


Рисунок 19 – Вводим некорректный выбор (символ)

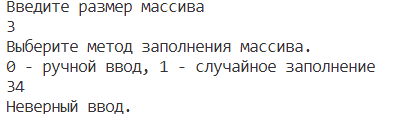


Рисунок 20 – Вводим некорректный выбор (отличный от 0 и 1)

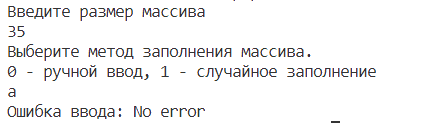


Рисунок 21 ‎– Вводим некорректный выбор (символ)

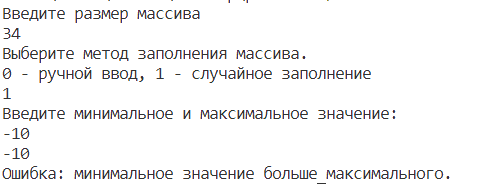


Рисунок 22 – Выбираем метод рандомного заполнения (вводим одинаковые значения)

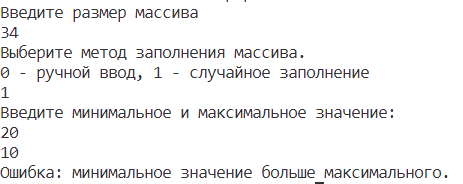


Рисунок 23 – Выбираем метод рандомного заполнения (минимум > максимума)

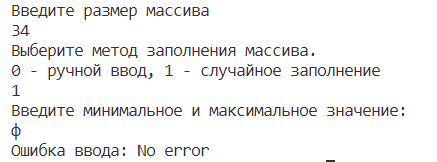


Рисунок 24 – Выбираем метод рандомного заполнения (вместо числа вводим букву)

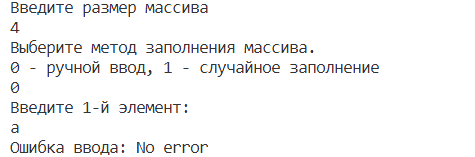


Рисунок 25 – Выбираем метод ручного заполнения (вместо числа вводим символ)

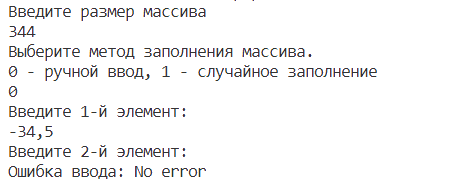


Рисунок 26 – Выбираем метод ручного заполнения (вместо int вводим double)

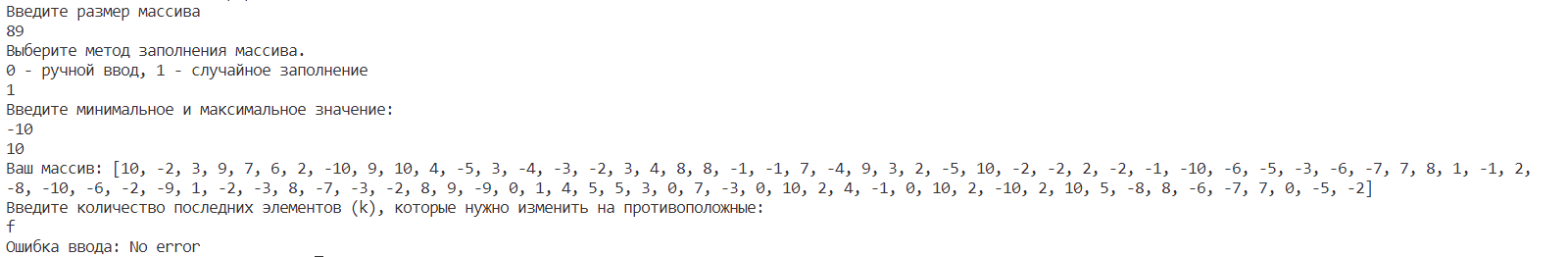


Рисунок 27 – Очередная проверка функции input()

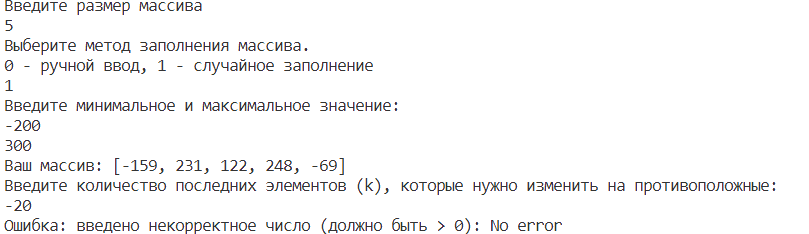


Рисунок 28 – Проверка функции pozitiv\_input()

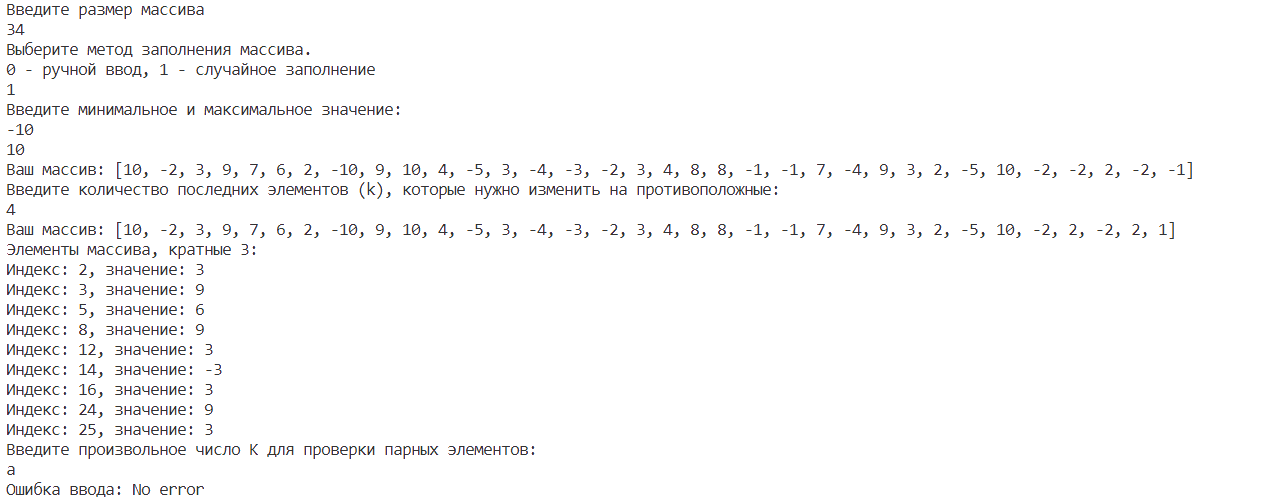


Рисунок 29 – Проверка функции input()

1.5 Отметка о выполнение программы

Отметка о выполнение задания в GitHub (Рисунок 30).

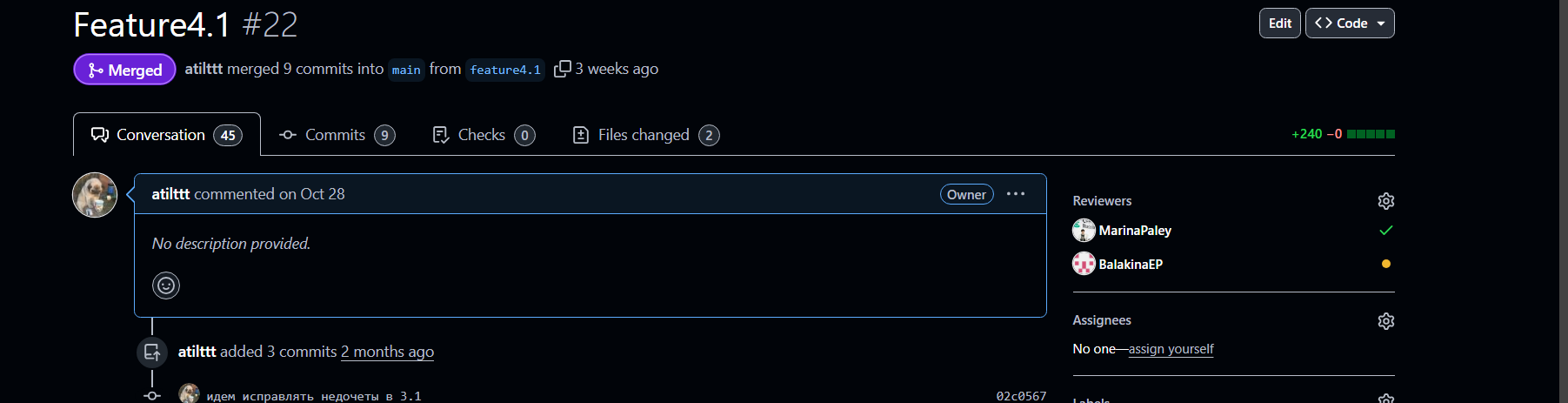


Рисунок 30 – Отметка о выполнение задания

1. 2 ЗАДАНИЕ 4.2

2.1 Формулировка задания

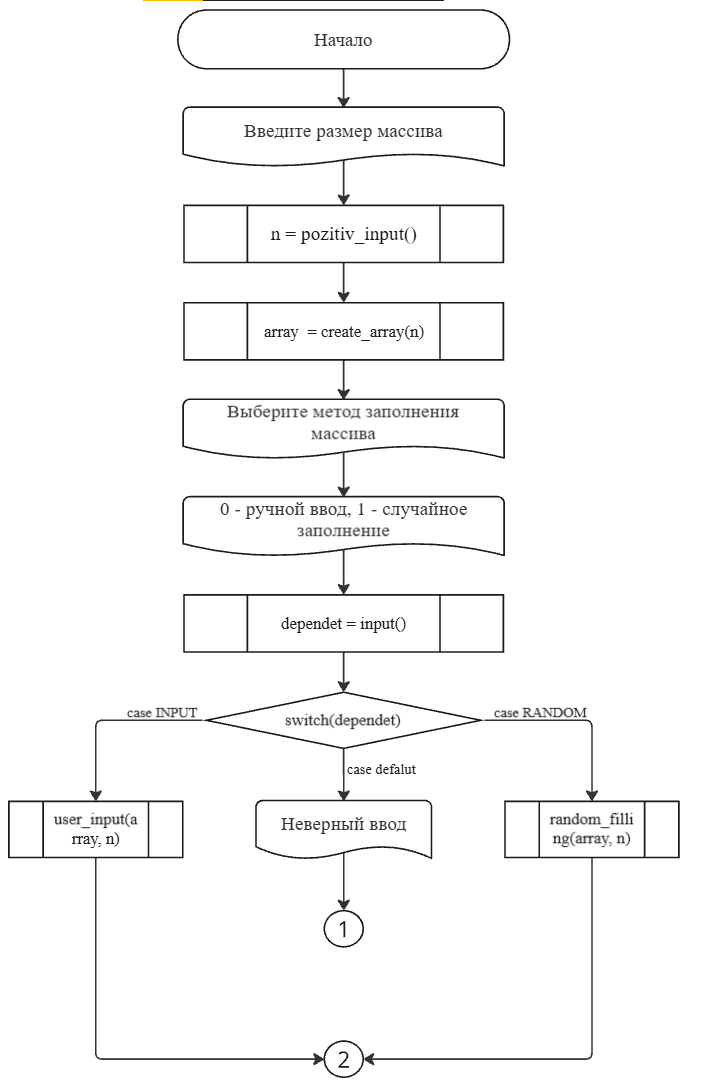
Создать одномерный массив из *n* целых чисел. Реализовать возможность заполнения массива, как случайными числами, так и с помощью клавиатуры по желанию пользователя. Пункты задания организовать в виде функций (методов) (). Вывести массив на экран. Составить блок-схему.

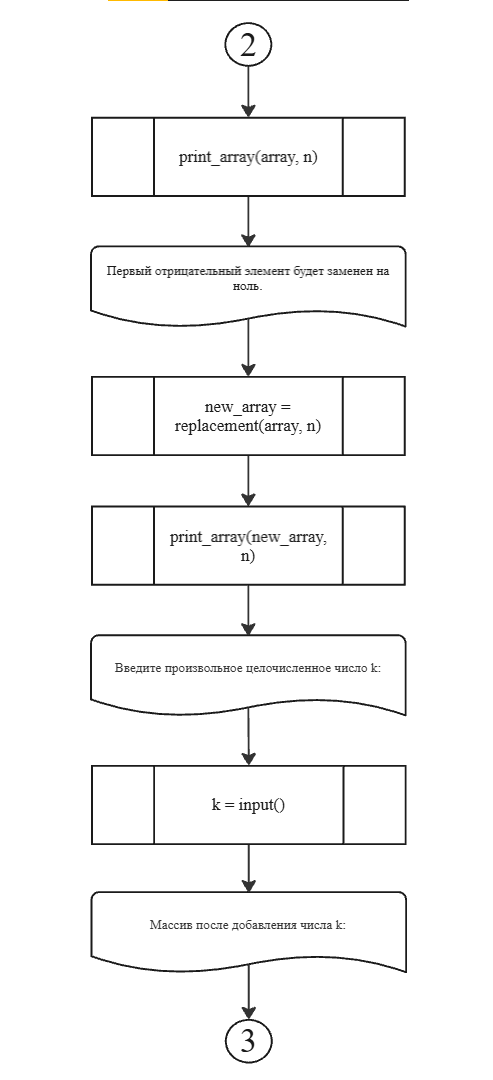
Таблица  – Исходные данные

|  |  |  |
| --- | --- | --- |
| **Вариант** | **Задача** | **Интервал** |
| **5** | 1. Заменить первый отрицательный элемент массива нулем. 2. Вставить число К после всех элементов, кратных своему номеру. 3. Из элементов массива D сформировать массив A той же размерности по правилу: если номер четный, то значение элемента находится по формуле Ai = (i-1)\*Di, а если нечетный, то по формуле Ai = Di\*i\*2. | [-100;100] |

2.2 Блок-схема основного алгоритма

Блок-схема основного алгоритма представлена ниже (Рисунок 31). Блок-схемы функций расчета (Рисунок 32, Рисунок 33, Рисунок 34, Рисунок 35, Рисунок 36, Рисунок 37, Рисунок 38, Рисунок 39, Рисунок 40, Рисунок 41, Рисунок 42, Рисунок 43).





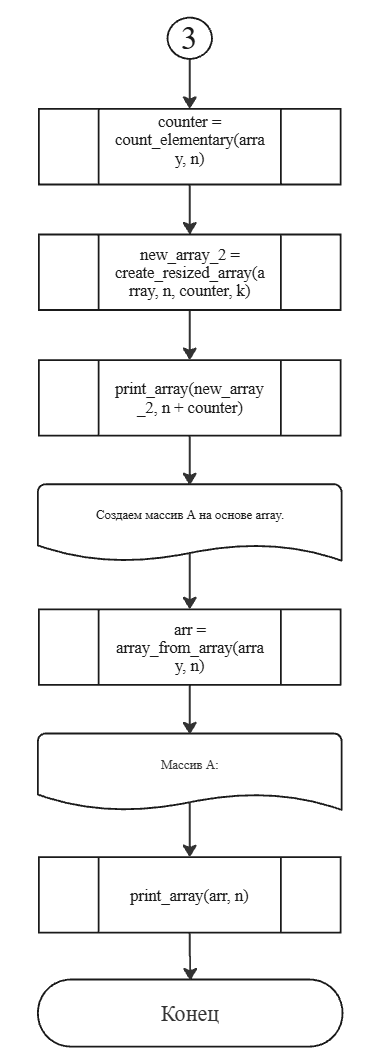


Рисунок  – Блок-схема основного алгоритма

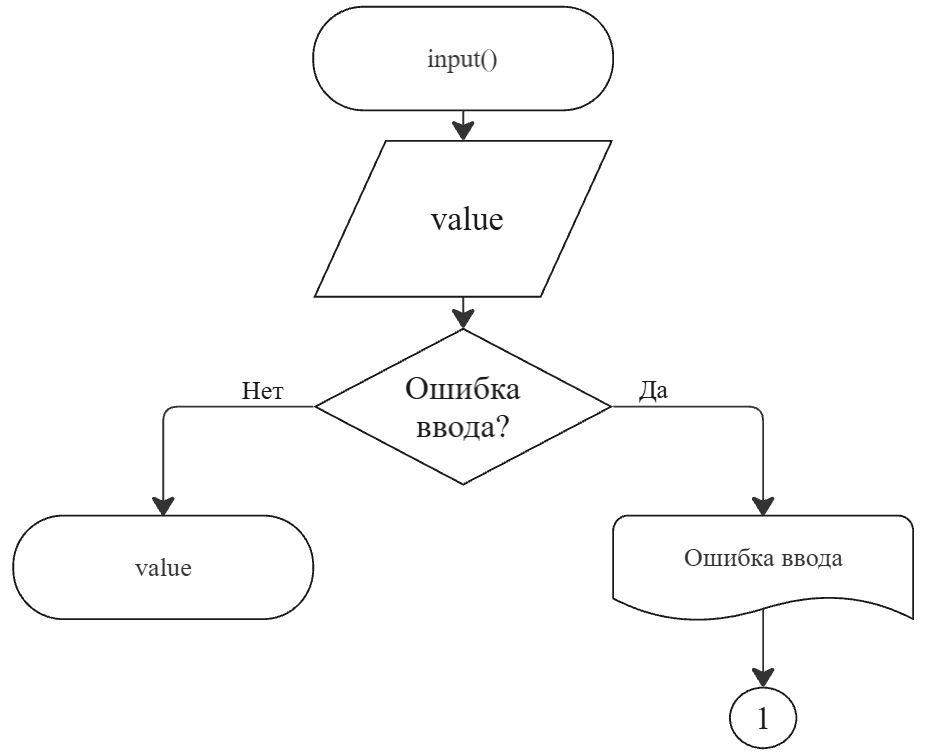


Рисунок  – Блок-схема функции input()

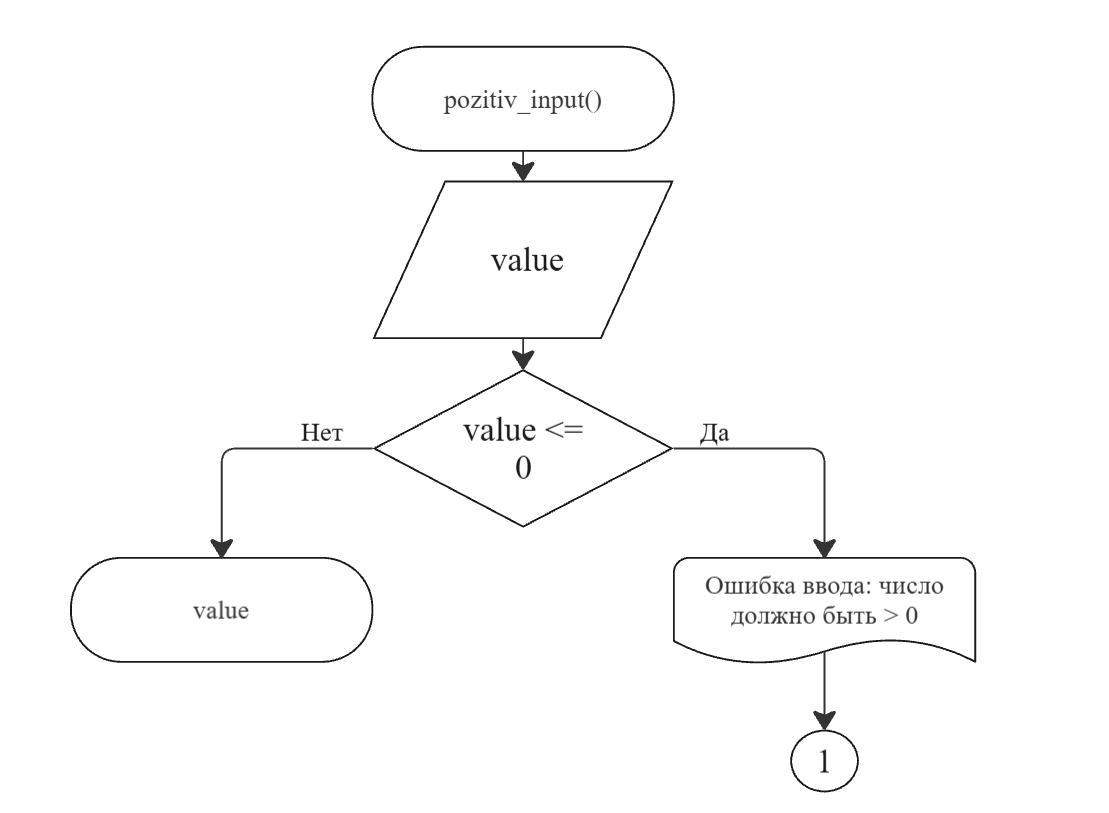


Рисунок  – Блок-схема функции pozitiv\_input()

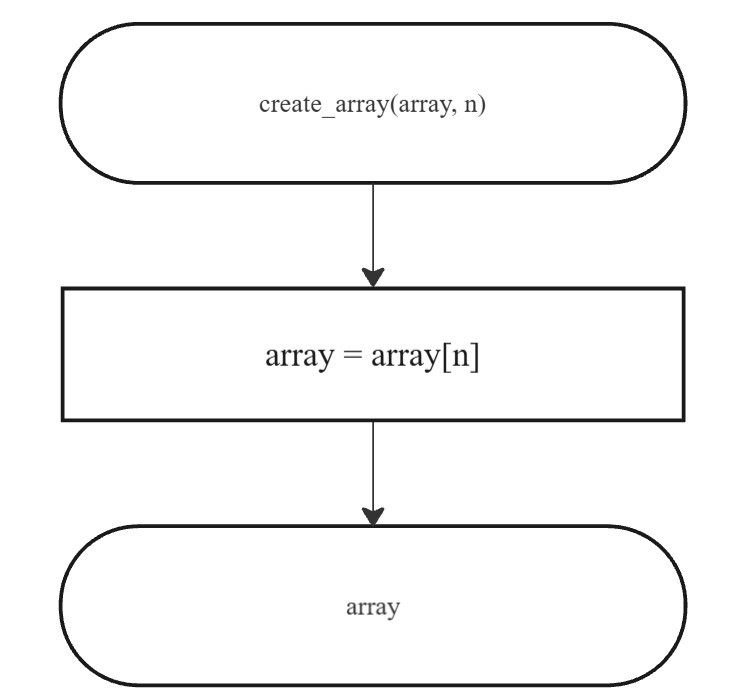


Рисунок 34 – Блок-схема функции create\_array(array, n)

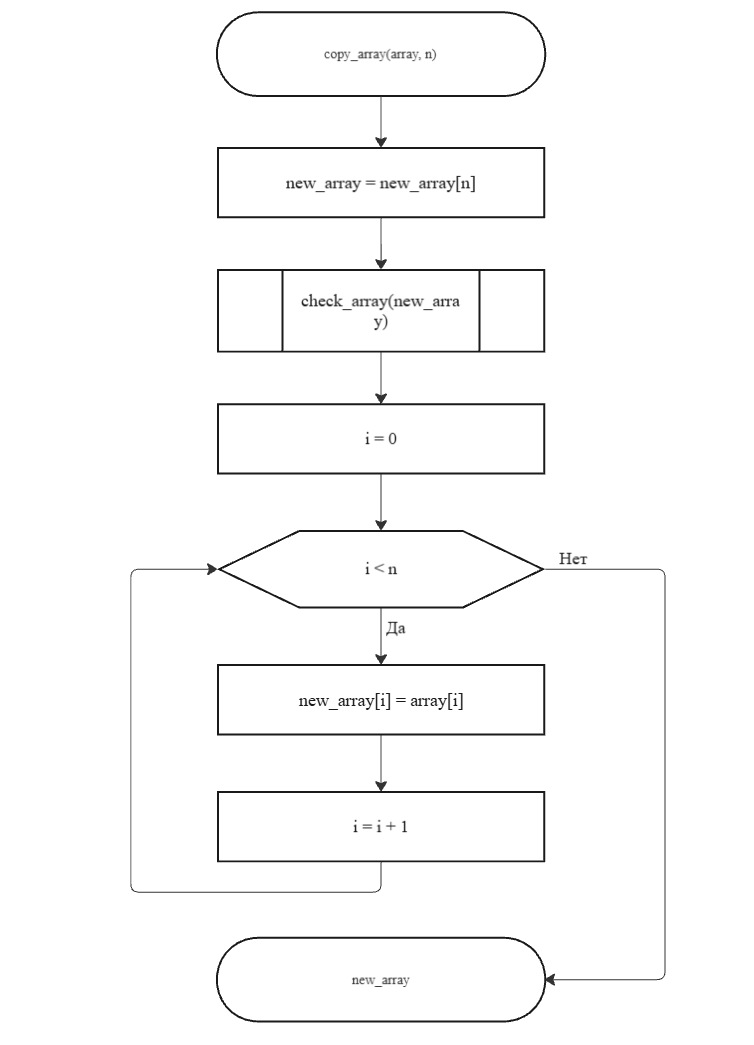


Рисунок  – Блок-схема функции copy\_array(array, n)

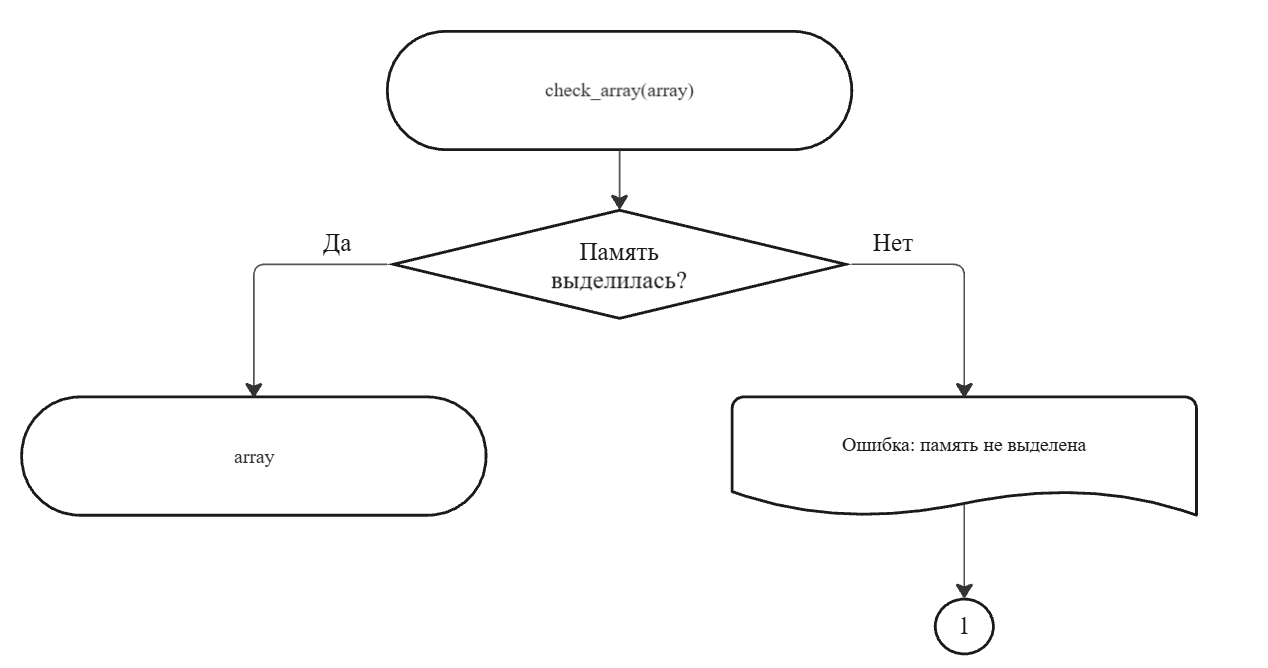


Рисунок  – Блок-схема функции check\_array(array)

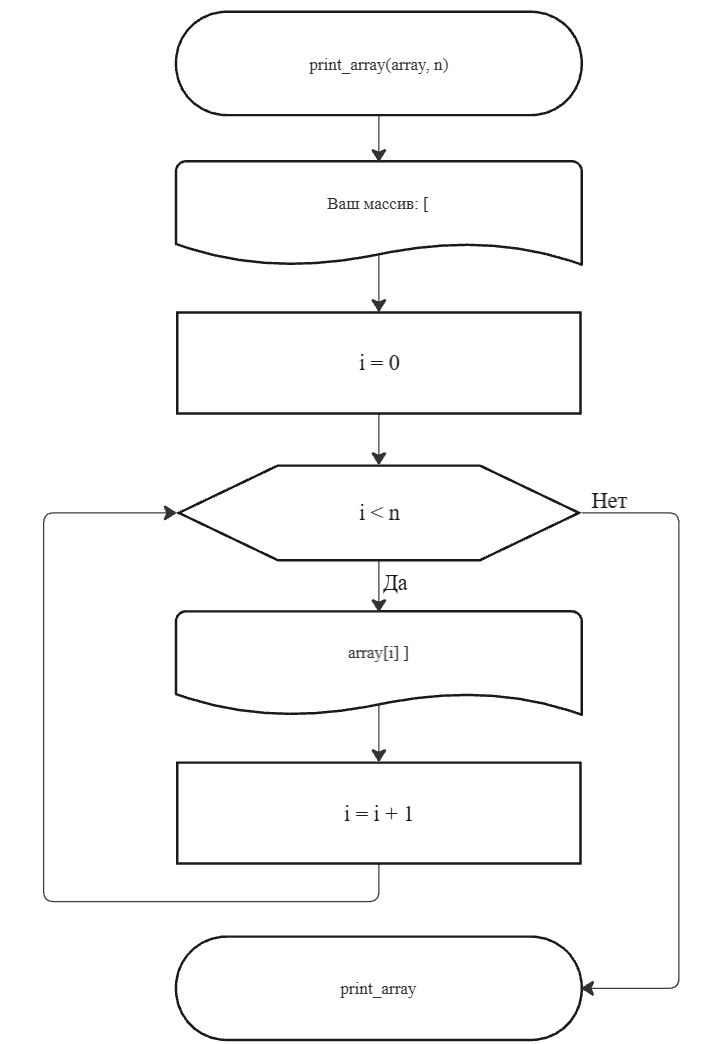


Рисунок 37 – Блок-схема функции print\_array(array, n)

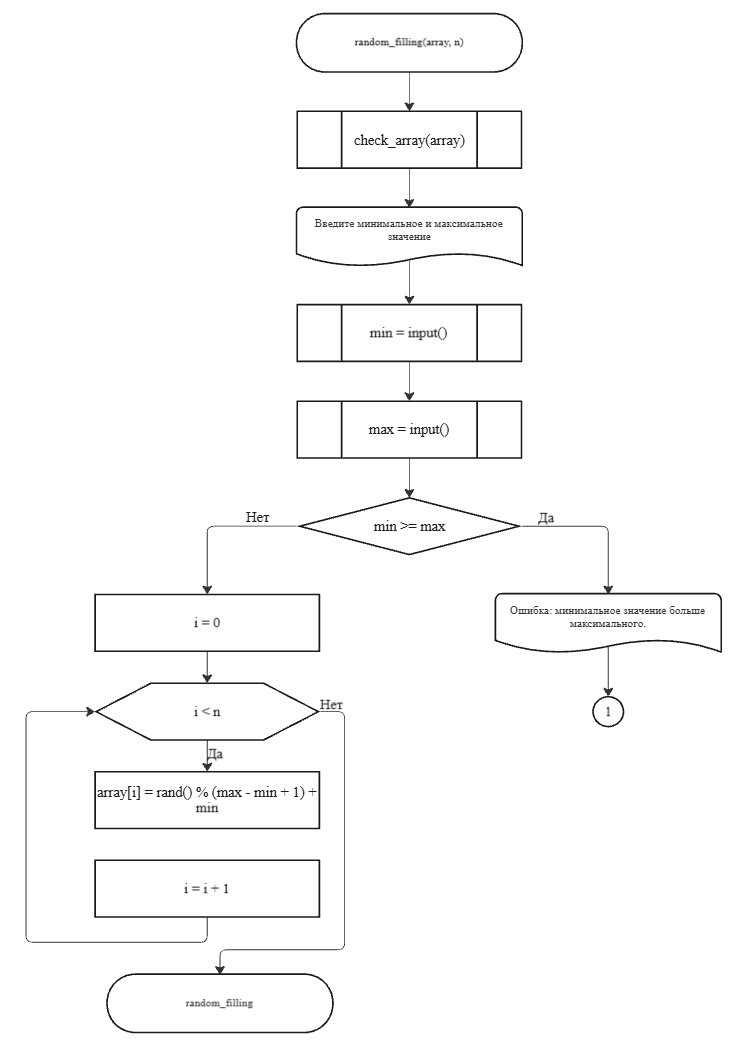


Рисунок 38 – Блок-схема функции random\_filling(array, n)

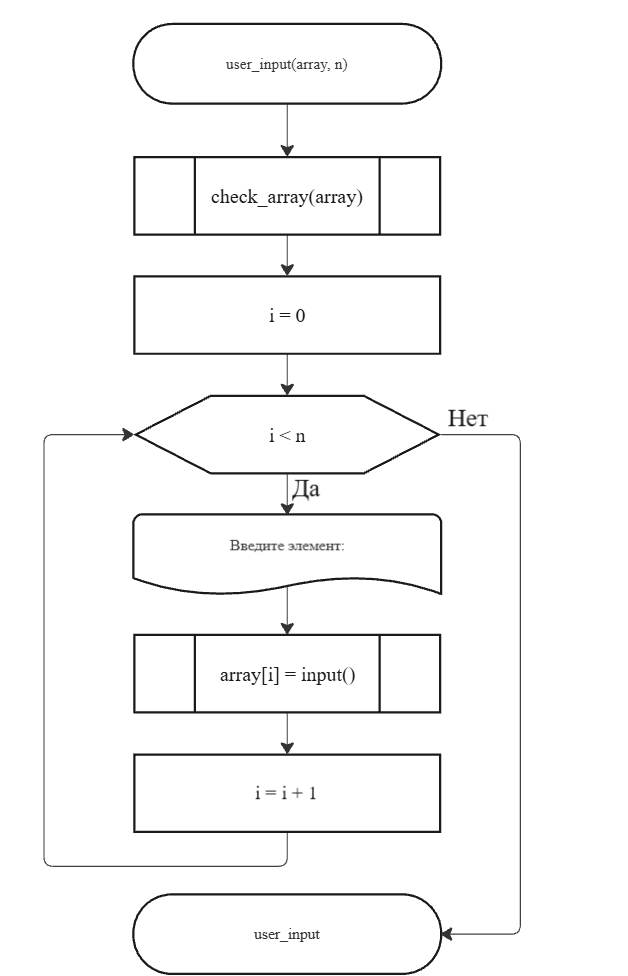


Рисунок 39 – Блок-схема функции input\_filling(array, n)

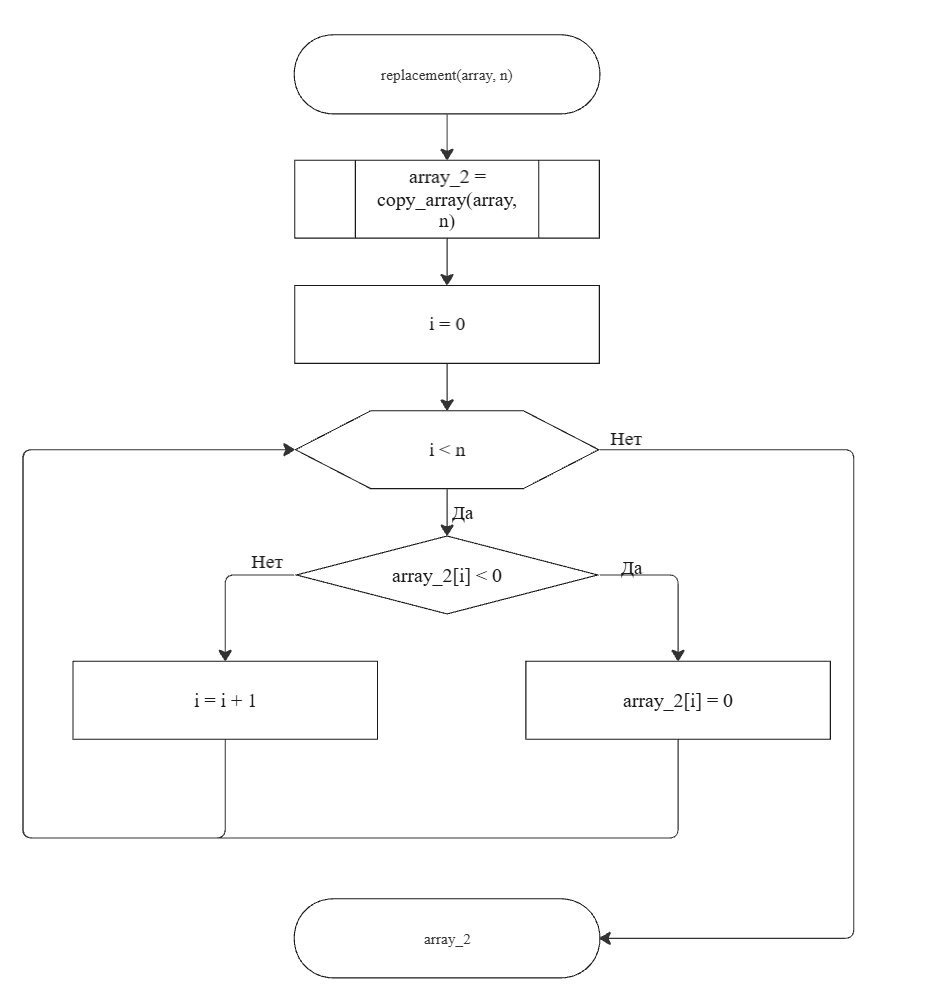


Рисунок 40 – Блок-схема функции replacement(array, n)

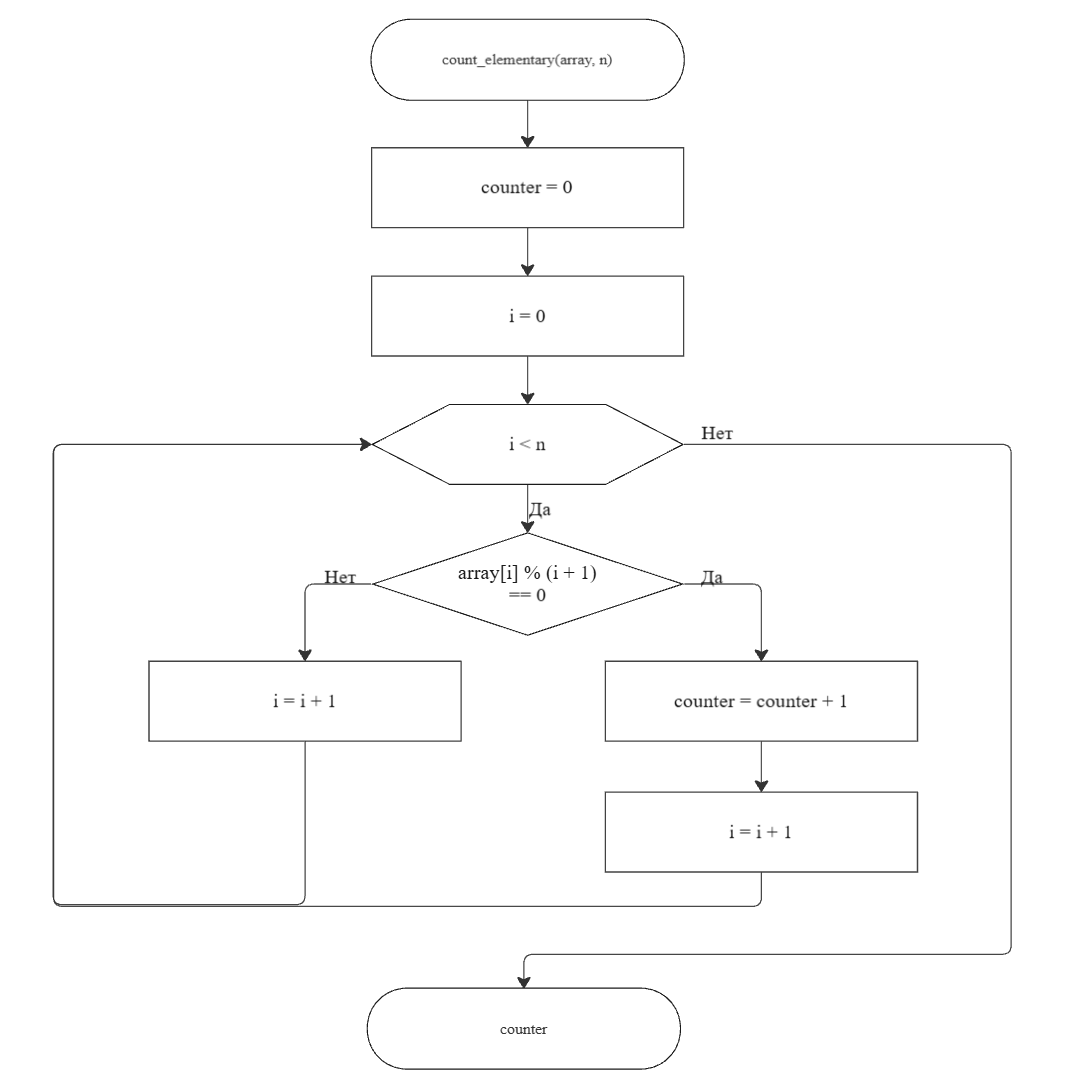


Рисунок 41 – Блок-схема count\_elementary(array, n)

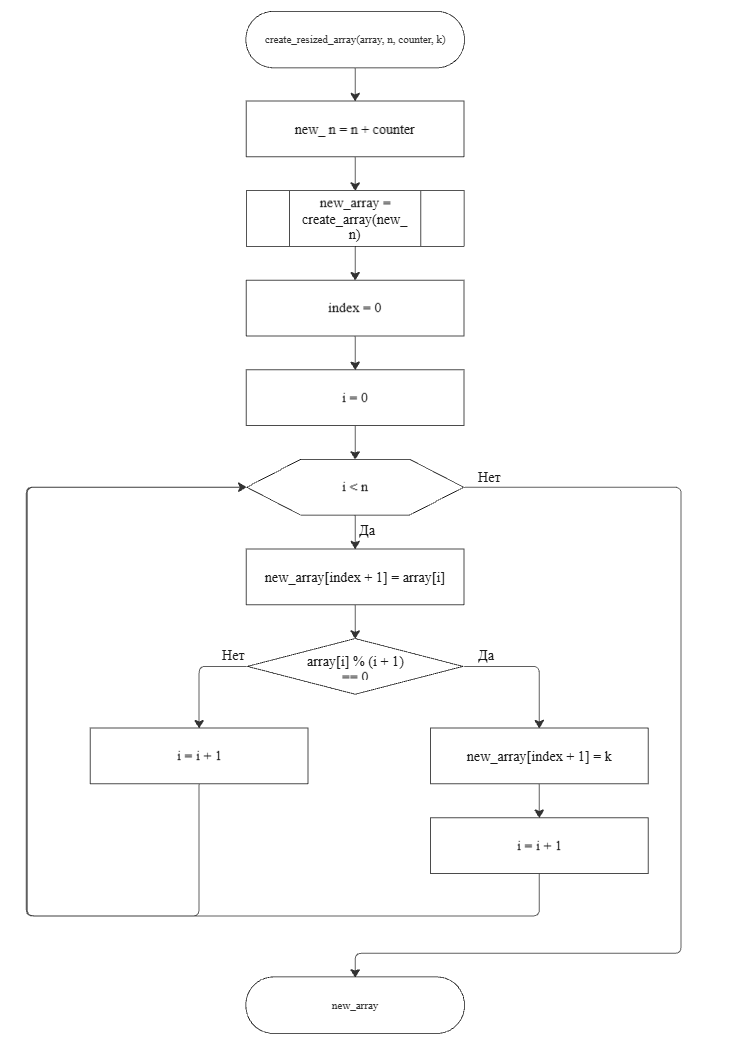


Рисунок 42 – Блок-схема функции create\_resized\_array(array, n, counter, k)

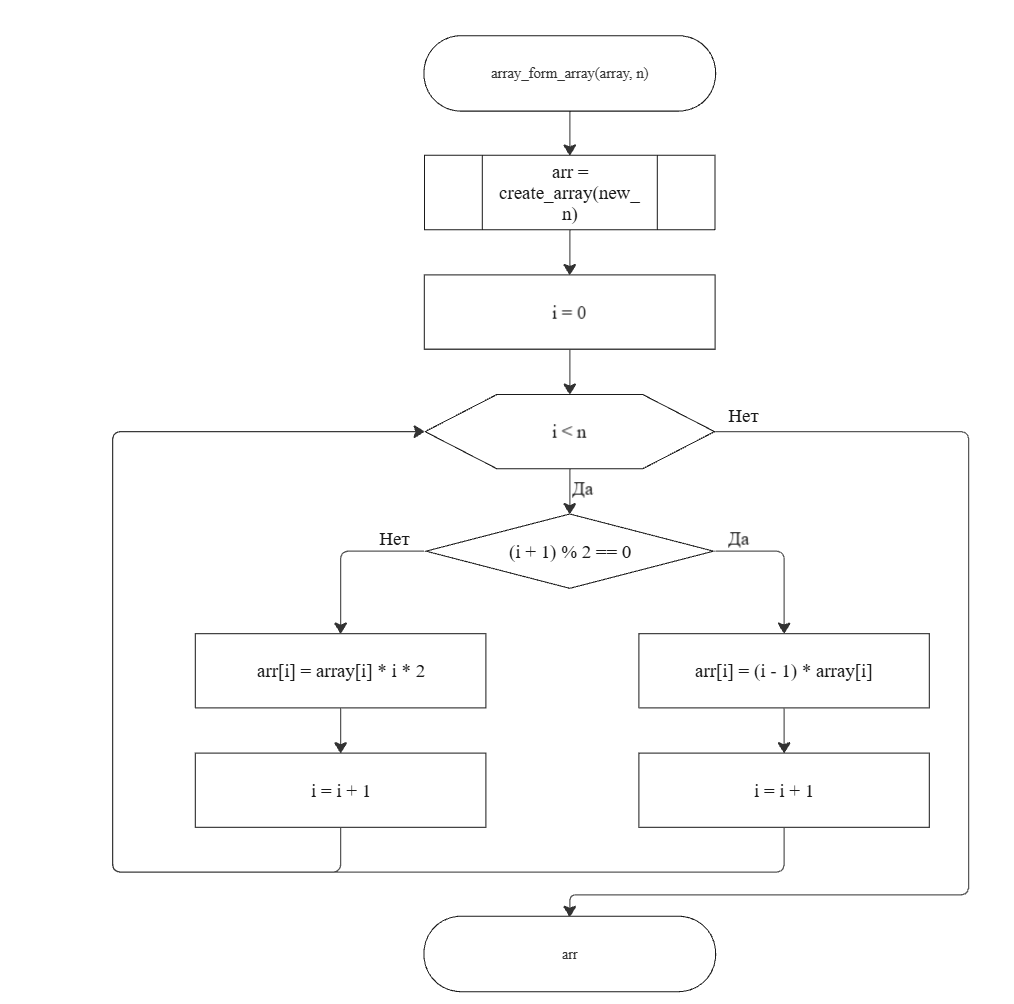


Рисунок 43 – Блок-схема функции array\_form\_array(array, n)

2.3 Текст программы на языке С

#include <stdio.h>

#include <errno.h>

#include <stdlib.h>

#include <math.h>

#include <time.h>

#include <locale.h>

#include <windows.h>

/\*\*

\* @brief функция проверяет пользовательский ввод

\* @return возвращает ошибку в случае, если пользователь ввел некоректное значение

\*/

int input(void);

/\*\*

\* @brief функция проверяет ввел ли пользователь положительное число

\* @return возвращает ошибку в случае, если число отрицательное

\*/

int pozitiv\_input(void);

/\*\*

\* @brief функция, создающая массив

\* @param n переменная, отвечающая за размер массива

\* @return созданный массив

\*/

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

/\*\*

\* @brief функция, копирающая массив

\* @param \*array указатель на массив (эквивалентно array[])

\* @param n переменная, отвечающая за размер массива

\* @return скопированный массив

\*/

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

/\*\*

\* @brief функция проверяющая значение адресса

\* @param \*array - указатель на массив (эквивалентно array[])

\*/

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

/\*\*

\* @param INPUT заполнение пользователем

\* @param RANDOM заполнение рандомными числами

\*/

enum choice {

INPUT\_TYPE = 2,

RANDOM = 1

};

/\*\*

\* @brief функция, выполняющая заполнение массива рандомными числами

\* @param array указатель на массив (эквивалентно array[])

\* @param n переменная, отвечающая за размер массива

\*/

void random\_filling(int \*array, const size\_t n);

/\*\*

\* @brief функция, выполняюзая заполнение массива с помощью пользовательского ввода

\* @param array указатель на массив (эквивалентно array[])

\* @param n переменная, отвечающая за размер массива

\*/

void user\_input\_filling(int \*array, const size\_t n);

/\*\*

\* @brief функция, которая выводит массив

\* @param array указатель на массив (эквивалентно array[])

\* @param n переменная, отвечающая за размер массива

\*/

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

/\*\*

\* @brief функция создает новый массив на основе прошлого

\* @param arrey указатель на массив (эквивалентно array[])

\* @param n переменная, отвечающая за размер массива

\*/

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

/\*\*

\* @brief функция находит первый отрицательный элемент в массиве и меняет его на 0

\* @param array указатель на массив (эквивалентно array[])

\* @param n переменная, отвечающая за размер массива

\*/

int\* replacement(int \*array, const size\_t n);

/\*\*

\* @brief считает элементы массива, которые кратны своему номеру

\* @param \*array указатель на массив (эквивалентно array[])

\* @param n переменная, отвечающая за размер массива

\* @return возвращает кол-во таких элементов

\*/

size\_t count\_elementary(const int \*array, size\_t n);

/\*\*

\* @brief функция вставляет элемент k, после каждого элемента массива, кратного своему номеру

\* @param \*array указатель на массив (эквивалентно array[])

\* @param n переменная, отвечающая за размер массива

\* @param counter переменная-счетчик (хранит в себе кол-во элементов, кратных своему номеру)

\* @param k переменная, хранящая в себе произвольное значение, введенное пользователем, которое вставляется после каждого элемента, кратного своему номеру

\* @return массив с вставленным в него числом K

\*/

int\* create\_resized\_array(const int \*array, const size\_t n, const size\_t counter, const int k);

/\*\*

\* @brief функция релизует массив А на основе массива array

\* @param указатель на массив (эквивалентно array[])

\* @param n переменная, отвечающая за размер массива

\*/

int\* array\_from\_array(const int \*array, const size\_t n);

int main(void) {

setlocale(LC\_ALL, "Russian");

SetConsoleCP(65001);

SetConsoleOutputCP(65001);

printf("Введите размер массива\n");

size\_t n = pozitiv\_input();

int \*array = create\_array(n);

printf("Выберите метод заполнения массива\n");

printf("Метод %d - заполнение рандомными числами, Метод %d - заполнение с клавиатуры\n", RANDOM, INPUT\_TYPE);

int dependet = pozitiv\_input();

switch (dependet) {

case INPUT\_TYPE:

user\_input\_filling(array, n);

break;

case RANDOM:

random\_filling(array, n);

break;

default:

printf("Произошла ошибка: некорректный ввод.\n");

free(array);

exit(EXIT\_FAILURE);

}

print\_array(array, n);

printf("Первый отрицательный элемент будет заменен на ноль.\n");

int\* new\_array = replacement(array, n);

print\_array(new\_array, n);

free(new\_array);

printf("Введите произвольное целочисленное число k:\n");

const int k = input();

printf("Массив после добавления числа k:\n");\

size\_t counter = count\_elementary(array, n);

int \*new\_array\_2 = create\_resized\_array(array, n, counter, k);

print\_array(new\_array\_2, n + counter);

free(new\_array\_2);

printf("Создаем массив A на основе array.\n");

int \*arr = array\_from\_array(array, n);

printf("Массив A:\n");

print\_array(arr, n);

free(array);

free(arr);

return 0;

}

int input(void) {

int value = 0;

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

if (result != 1) {

perror("Ошибка ввода");

exit(EXIT\_FAILURE);

}

return value;

}

int pozitiv\_input(void) {

int value = input();

if (value <= 0) {

perror("Ошибка ввода: число должно быть > 0");

exit(EXIT\_FAILURE);

}

return value;

}

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

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

check\_array(array);

return array;

}

int\* copy\_array(const int \*array, const size\_t n) {

check\_array(array);

int \*new\_array = create\_array(n);

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

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

}

return new\_array;

}

void check\_array(const int \*array) {

if (array == NULL) {

perror("Ошибка выделения памяти");

exit(EXIT\_FAILURE);

}

}

void random\_filling(int \*array, const size\_t n) {

check\_array(array);

printf("Введите минимальное и максимальное значения:\n");

int min = input();

int max = input();

if (min >= max) {

printf("Ошибка: минимальное значение больше максимального.\n");

exit(EXIT\_FAILURE);

}

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

array[i] = rand() % (max - min + 1) + min;

}

}

void user\_input\_filling(int \*array, const size\_t n) {

check\_array(array);

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

printf("Введите элемент %zu:\n", i + 1);

array[i] = input();

}

}

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

check\_array(array);

printf("Ваш массив:\n[");

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

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

}

printf("]\n");

}

int\* replacement(int \*array, const size\_t n) {

int \*array\_2 = copy\_array(array, n);

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

if (array\_2[i] < 0) {

array\_2[i] = 0;

break;

}

}

return array\_2;

}

size\_t count\_elementary(const int \*array, size\_t n) {

size\_t counter = 0;

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

if (array[i] % (i + 1) == 0) {

++counter;

}

}

return counter;

}

int\* create\_resized\_array(const int \*array, const size\_t n, const size\_t counter, const int k) {

size\_t new\_n = n + counter;

int \*new\_array = create\_array(new\_n);

size\_t index = 0;

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

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

if (array[i] % (i + 1) == 0) {

new\_array[index++] = k;

}

}

return new\_array;

}

int\* array\_from\_array(const int \*array, const size\_t n) {

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

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

if ((i + 1) % 2 == 0) {

arr[i] = (i - 1) \* array[i];

} else {

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

}

}

return arr;

}

2.4 Результат выполнения программы

Ниже представлен результат выполнения программы (Рисунок 44, Рисунок 45). Также представлены тестовые примеры, которые потенциально могут сломать программу (Рисунок 46, Рисунок 47, Рисунок 48, Рисунок 49, Рисунок 50, Рисунок 51, Рисунок 52, Рисунок 53, Рисунок 54, Рисунок 55).

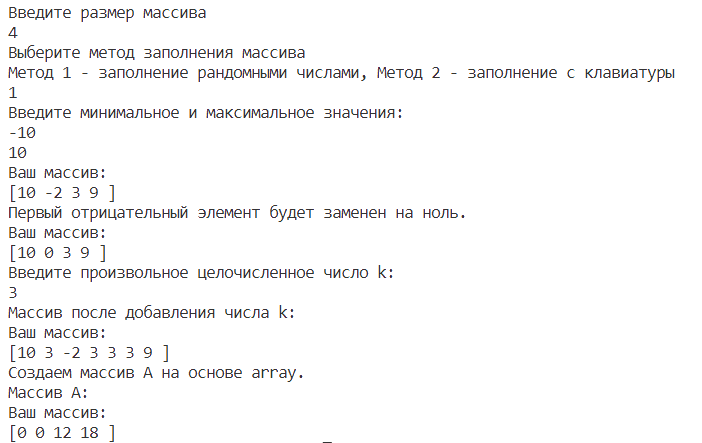


Рисунок  – Первый результат выполнения программы

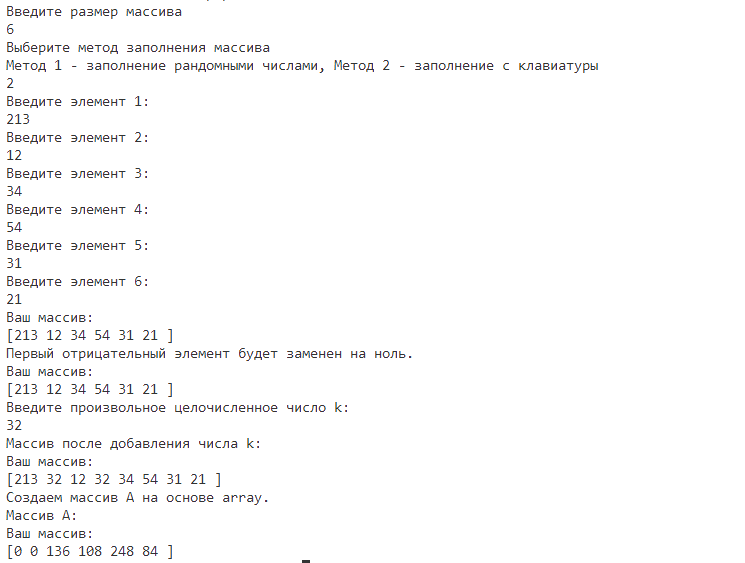


Рисунок  – Второй результат выполнения программы



Рисунок  – Проверка функции pozitiv\_input()



Рисунок – Проверка функции input()



Рисунок  – Очередная проверка функции input()

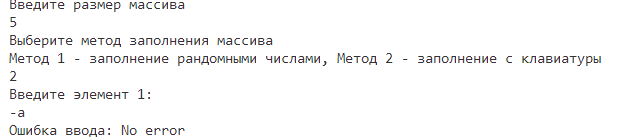


Рисунок – Вводим некорректный элемент массива (символ)

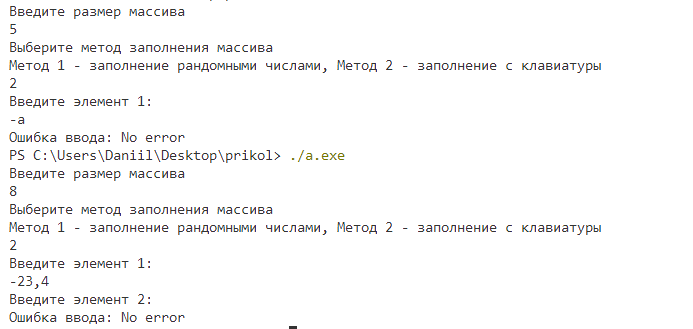


Рисунок  – Вводим некорректный элемент массива (double)

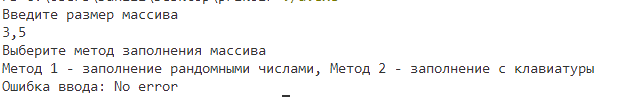


Рисунок  – Так же проверяем, что будет, если ввести double для размера массива

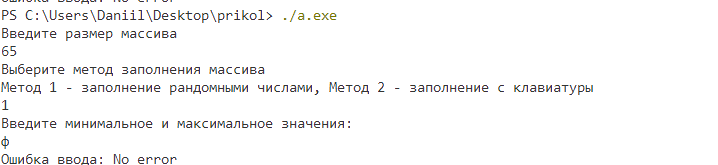


Рисунок  – В очередной раз проверяем функцию input()

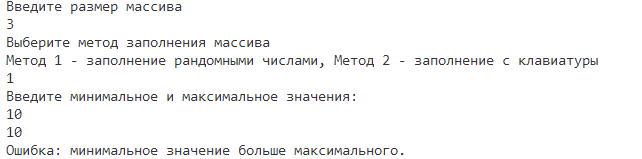


Рисунок  – Проверяем min = max

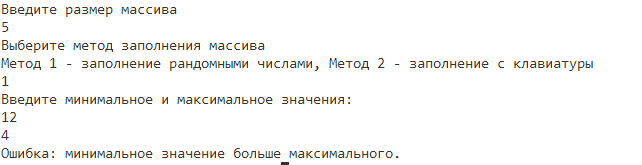


Рисунок  – Проверяем min > max

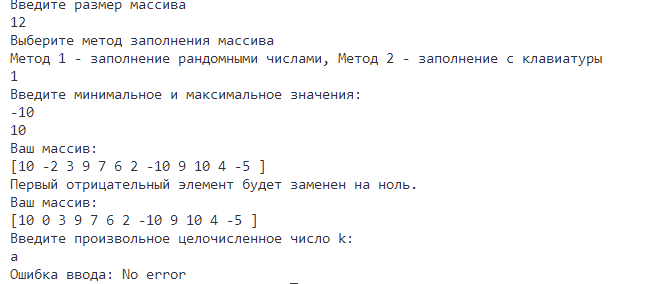


Рисунок – Очередная проверка функции input()

2.5 Отметка о выполнение задания

Отметка о выполнение задания в GitHub (Рисунок 56).

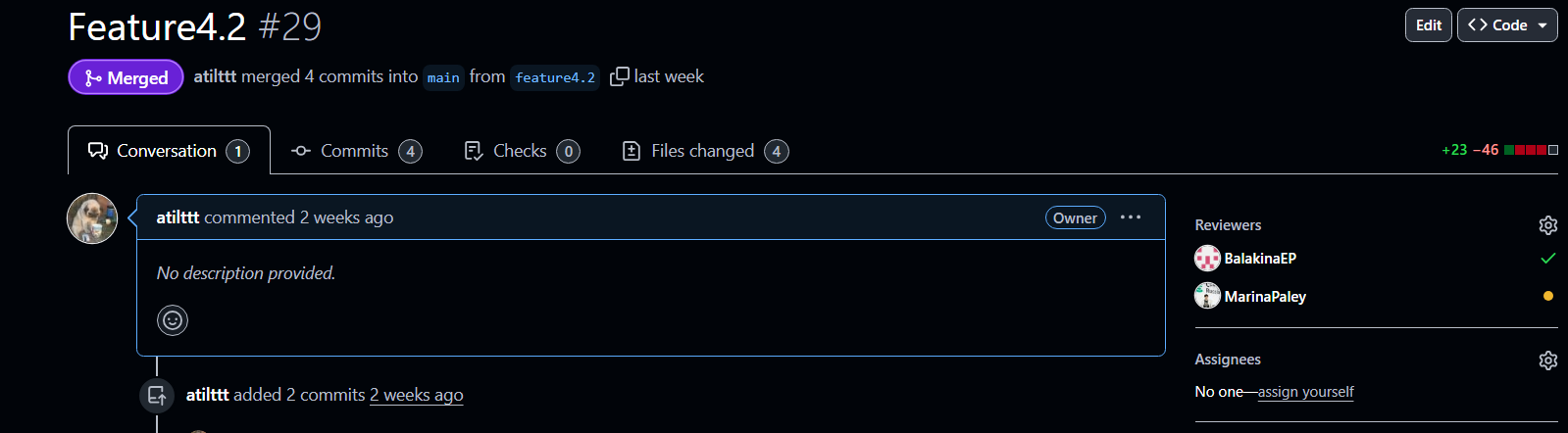


Рисунок  – Отметка о выполнение задания