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 10

Выполнил: ст. гр. ТКИ-142

Теканов Фёдор Андреевич

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

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

Москва 2024

1 формулировка задания

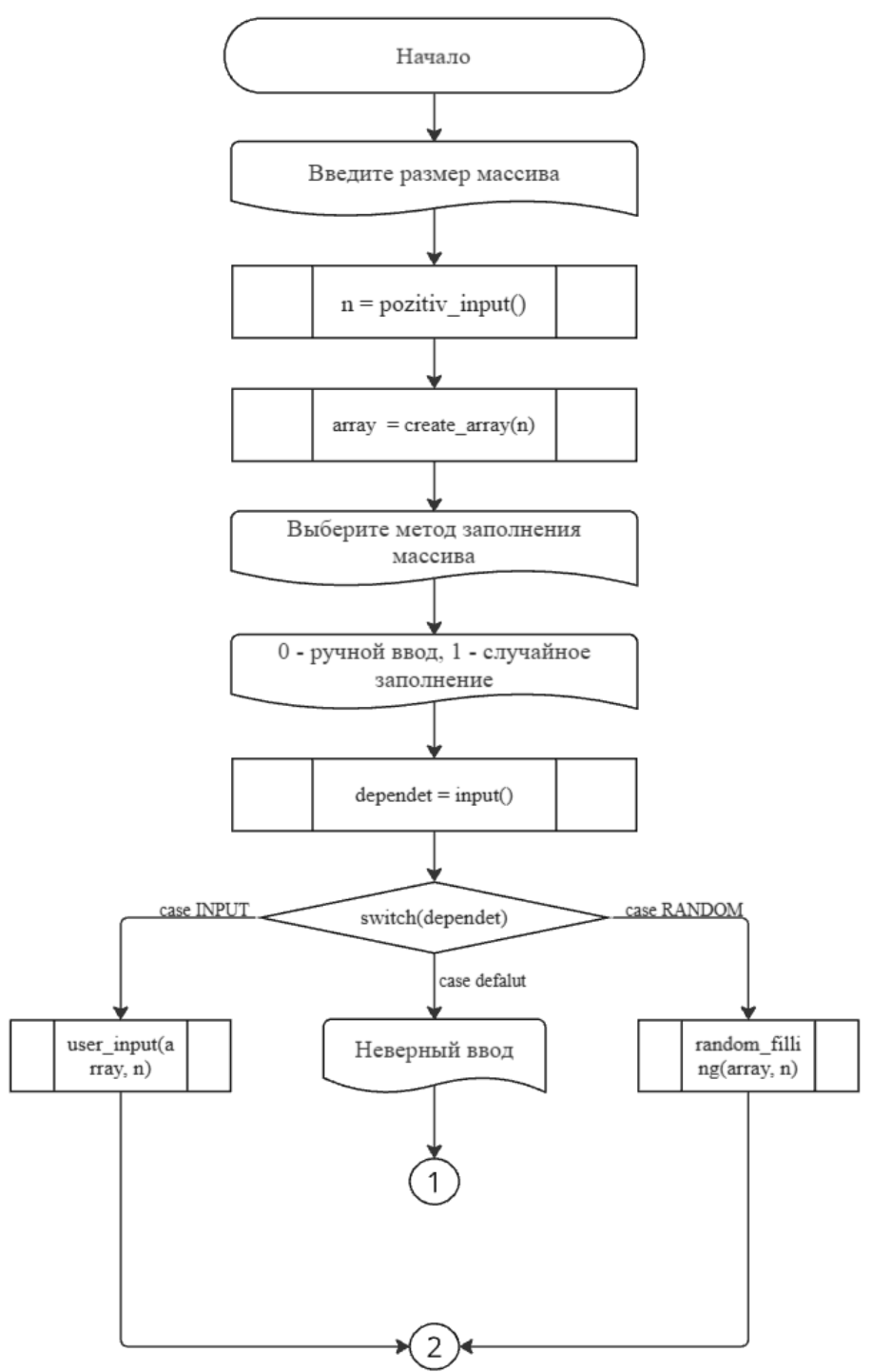
Создать одномерный массив из *n* целых чисел. Реализовать возможность заполнения массива, как случайными числами, так и с помощью клавиатуры по желанию пользователя. Пункты задания организовать в виде функций (методов). Составить блок-схему. (Таблица 1)

Таблица  – Исходные данные

|  |  |  |
| --- | --- | --- |
| **Вариант** | **Задача** | **Интервал** |
| **10** | 1. Найти сумму отрицательных элементов. 2. Найти количество тех элементов, значения которых положительны и не превосходят заданного числа А. 3. Найти номер последней пары соседних элементов с разными знаками. | [-100;200] |

2 блок-схема основного алгоритма

Блок-схема основного алгоритма представлена ниже (Рисунок 1). Блок-схемы функций расчета (Рисунок 2, Рисунок 3, Рисунок 4, Рисунок 5, Рисунок 6, Рисунок 7, Рисунок 8).



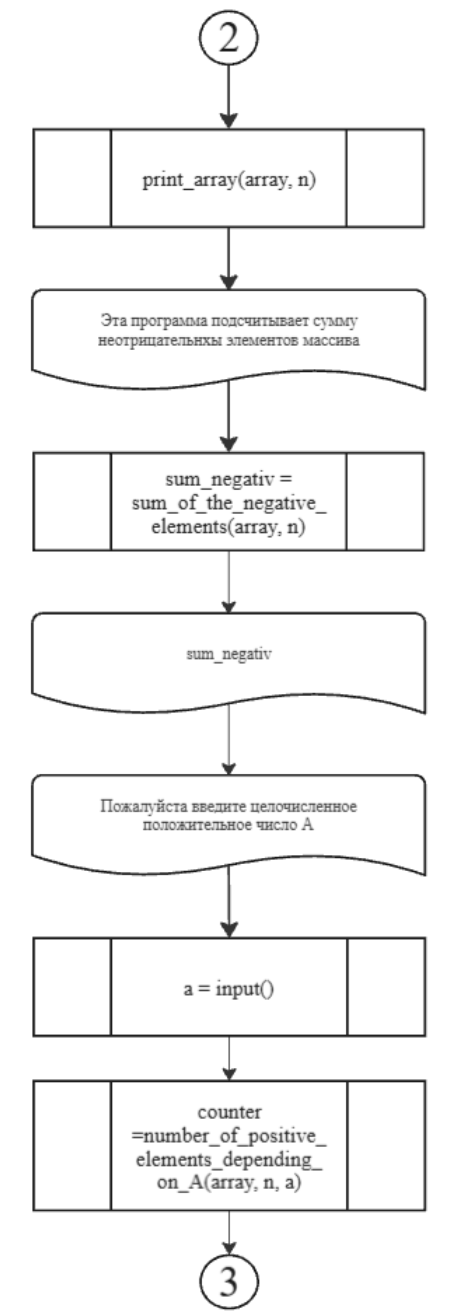




Рисунок – Блок-схема основного алгоритма

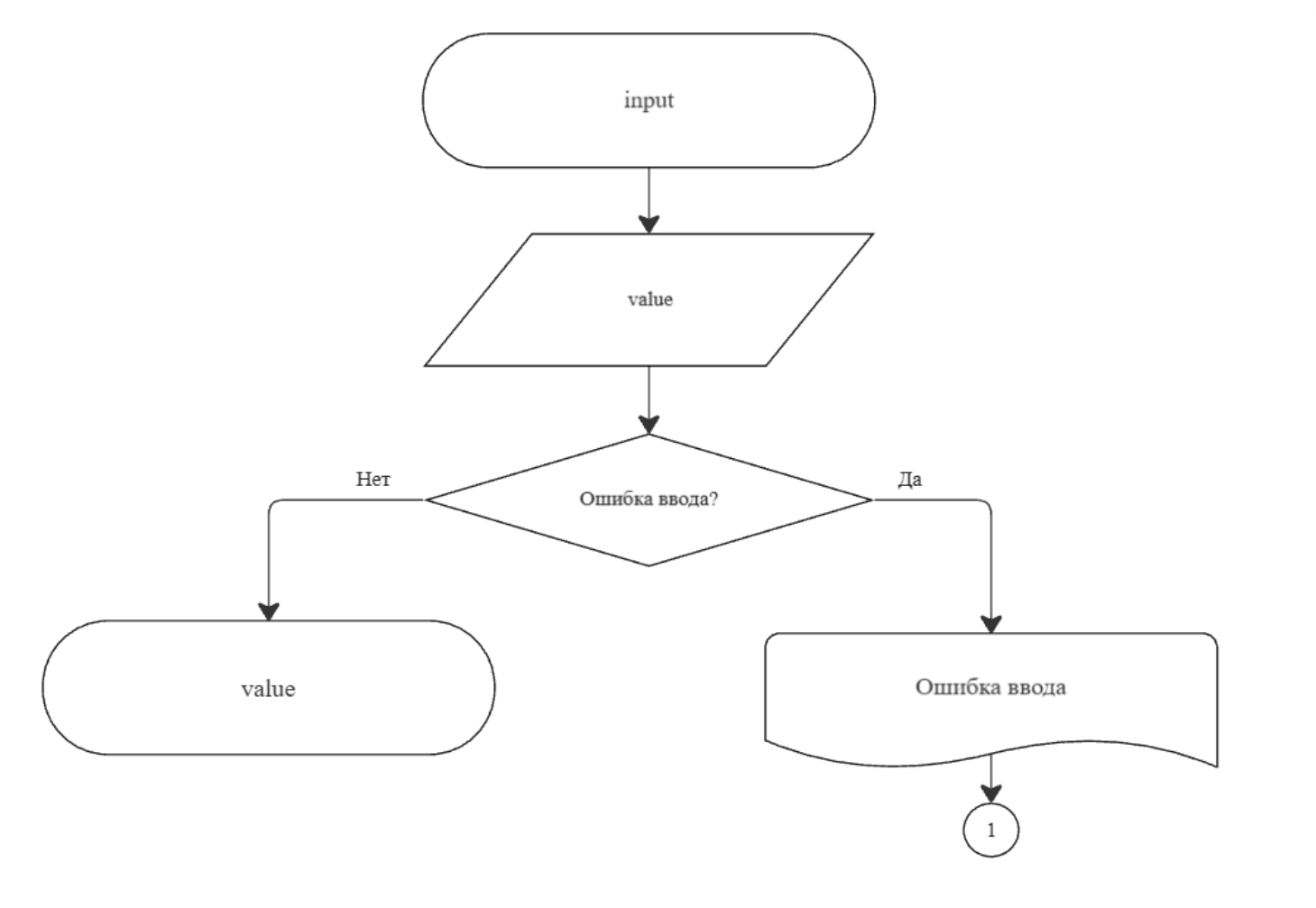


Рисунок  – Блок-схема функции input

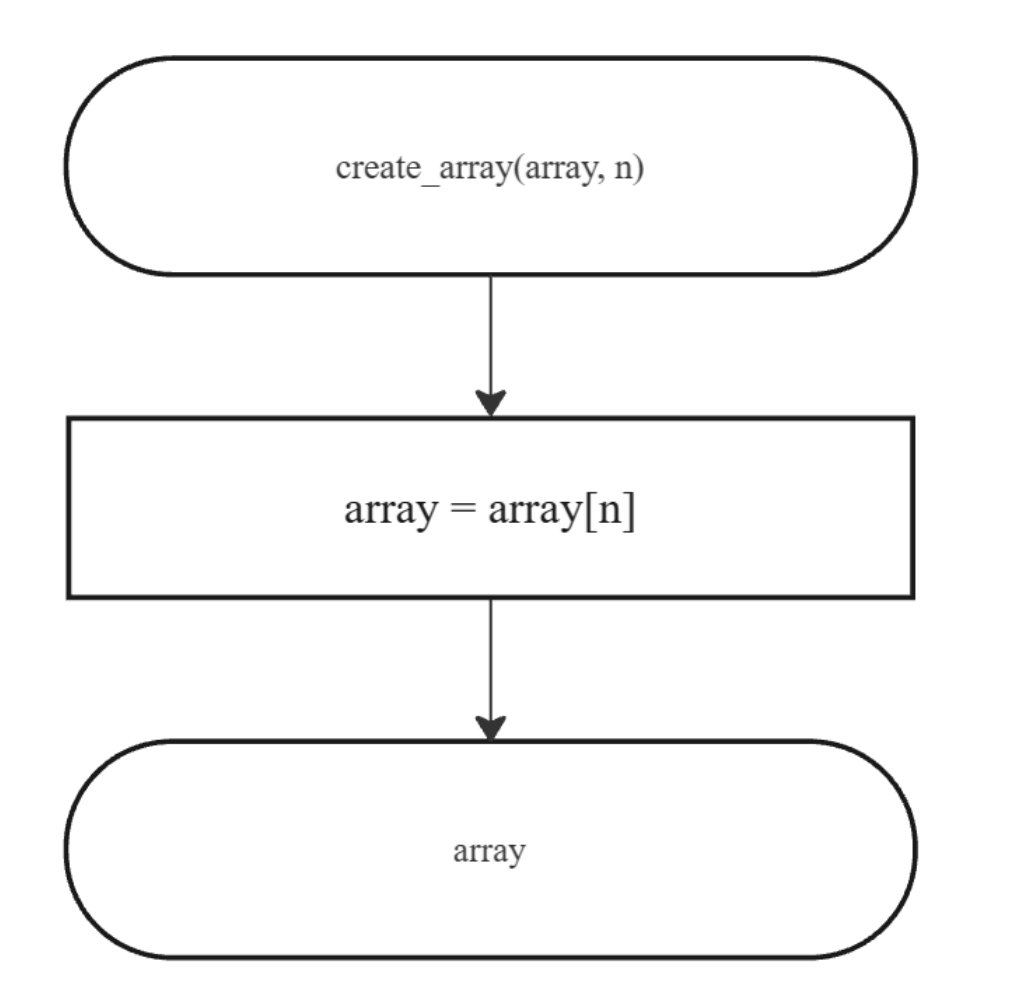


Рисунок 3 – Блок-схема функции create\_array(array, n)

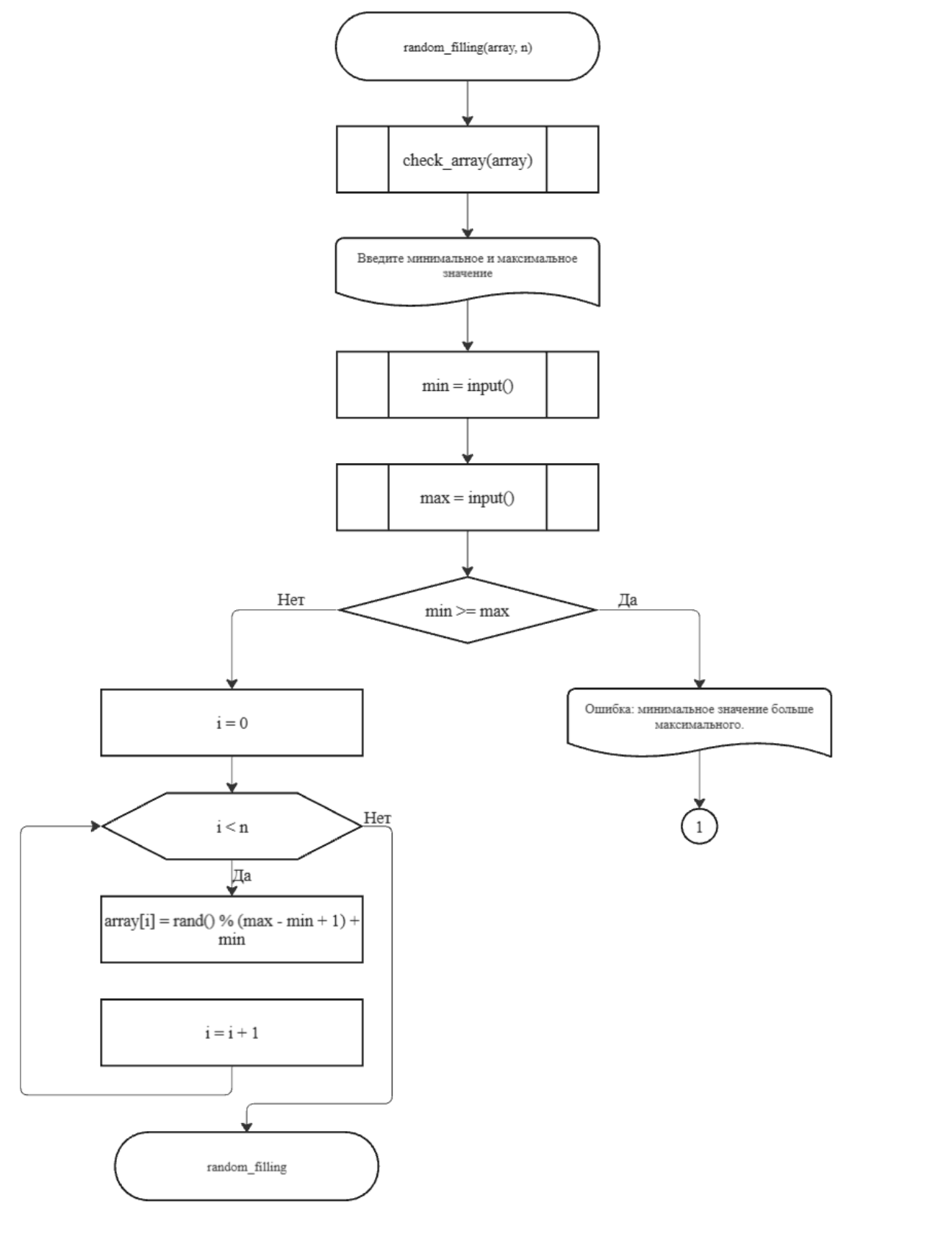


Рисунок  – Блок-схема функции random\_filling

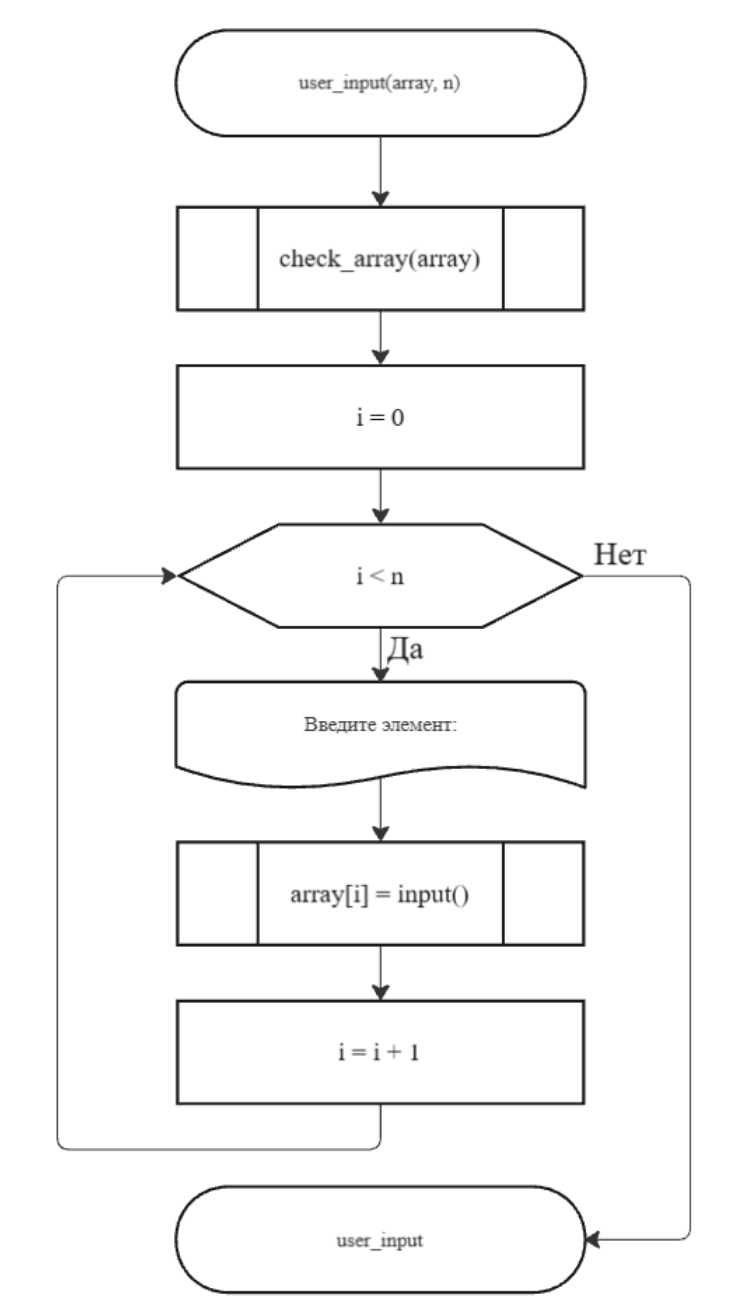


Рисунок  – блок-схема функции user\_input()

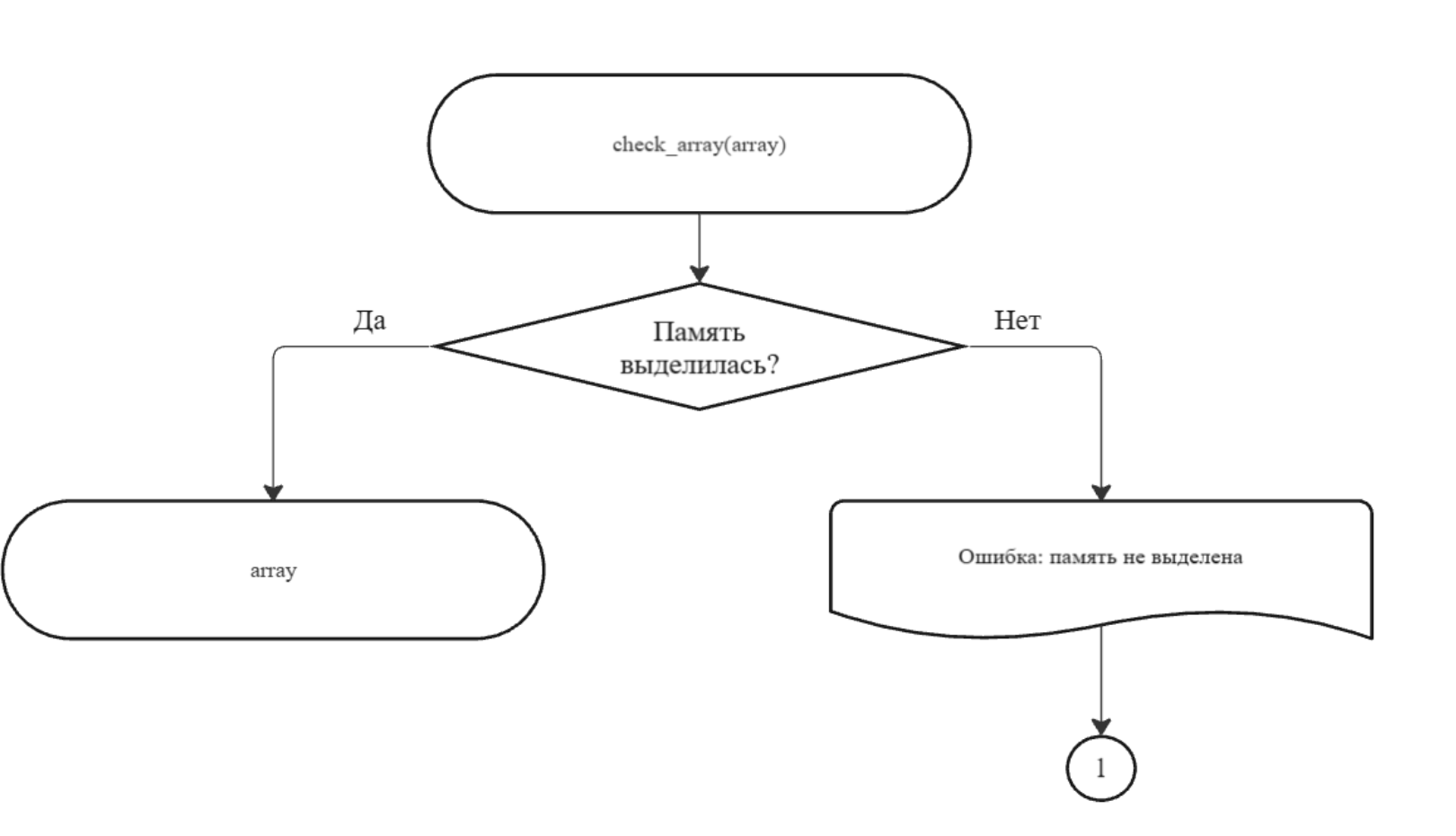


Рисунок – Блок-схема функции check\_array(array)

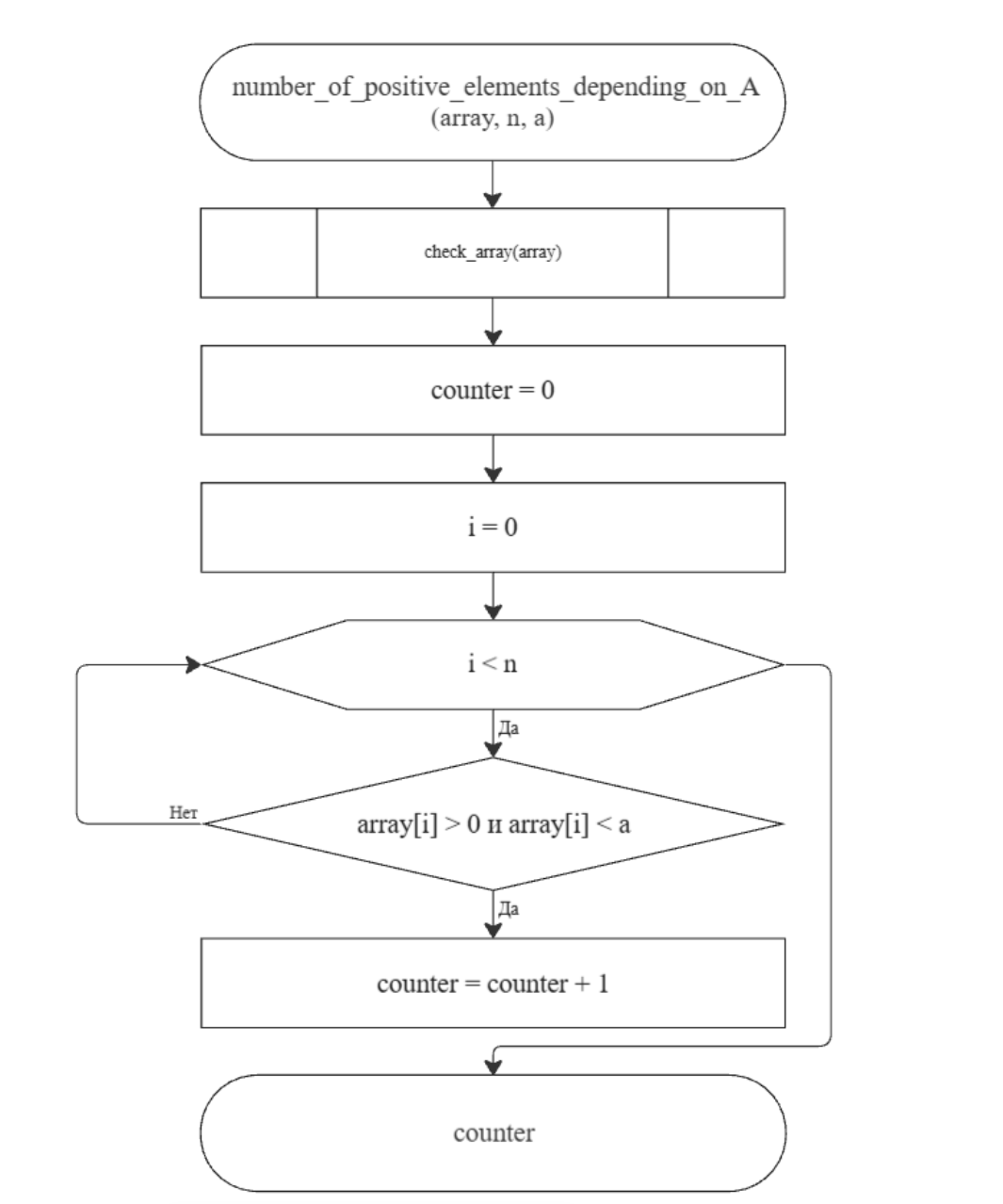


Рисунок 7 – Блок-схема функции number\_of\_positive\_elements\_depending\_on\_A(array, n, a)

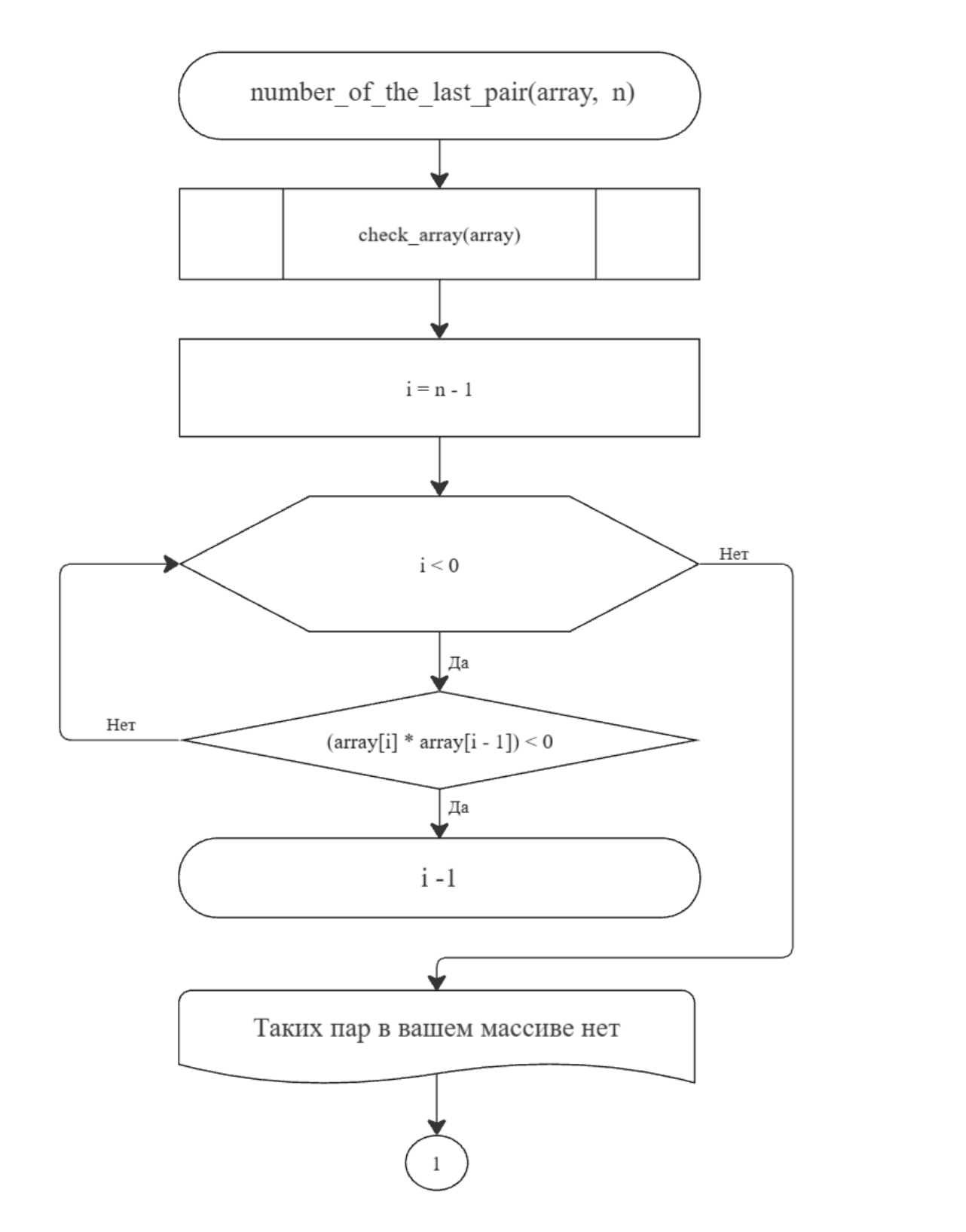


Рисунок 8 – блок-схема функции  number\_of\_the\_last\_pair(array, n)

3 код 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(const 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 функция, считающая сумму отрицательных элементов

\* @param \*array указатель на массив

\* @param n переменная, отвечающая за размер массива

\* @return возвращает рассчитанную сумму

\*/

int sum\_of\_the\_negative\_elements(const int \*array, const size\_t n);

/\*\*

\* @brief функция, считающая кол-во положительных элементов непревосходящих заданного числа

\* @param \*array указатель на массива

\* @param n переменная, отвечающая за размер массива

\* @param a заданное число

\* @return кол-во таких элементов

\*/

int number\_of\_positive\_elements\_depending\_on\_A(const int \*array, const size\_t n, const int a);

/\*\*

\* @brief функция, считающая кол-во пар с разными знаками

\* @param \*array указатель на массив

\* @param n переменная отвечающая за размер массива

\* @return возвращает кол-во таких пар

\*/

int pairs\_of\_elements(const int \*array, const size\_t n);

int number\_of\_the\_last\_pair(const int \*array, const size\_t n);

#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("%d - ручной ввод, %d - случайное заполнение\n", INPUT, RANDOM);

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("Эта программа подсчитывает сумму неотрицательнхы элементов массива\n");

int sum\_negativ = sum\_of\_the\_negative\_elements(array, n);

printf("Сумма таких элементов: %d\n", sum\_negativ);

printf("Пожалуйста введите целочисленное положительное число A\n");

int a = pozitiv\_input();

int counter = number\_of\_positive\_elements\_depending\_on\_A(array, n, a);

printf("Кол-во положительных элементов массива < A: %d\n", counter);

printf("Так же эта программа находит номер последней пары, в который элементы имеют разные знаки");

int pairs = number\_of\_the\_last\_pair(array, n);

printf("Номер этой пары: %d", pairs);

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) {

check\_array(array);

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) {

check\_array(array);

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) {

check\_array(array);

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 sum\_of\_the\_negative\_elements(const int \*array, const size\_t n)

{

check\_array(array);

int sum = 0;

int nema = 0;

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

{

if (array[i] < 0)

{

sum += array[i];

}

else

{

nema++;

}

}

if (nema == n)

{

printf("В вашем массиве нет отрицательных элементов\n");

}

return sum;

}

int number\_of\_positive\_elements\_depending\_on\_A(const int \*array, const size\_t n, const int a)

{

check\_array(array);

int counter = 0;

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

{

if (array[i] > 0 && array[i] < a)

counter++;

}

if (counter == 0)

{

printf("Таких элементов не нашлось\n");

}

return counter;

}

int pairs\_of\_elements(const int \*array, const size\_t n)

{

check\_array(array);

int counter = 0;

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

{

if ((array[i] \* array[i + 1])< 0 )

counter++;

}

if (counter == 0)

printf("Таких пар в вашем массиве нет\n");

return counter;

}

int number\_of\_the\_last\_pair(const int \*array, const size\_t n)

{

check\_array(array);

for (size\_t i = n - 1; i > 0; i--)

{

if ((array[i] \* array[i - 1]) < 0)

return i - 1;

}

printf("Таких пар в вашем массиве нет\n");

return -1;

}

4 результат выполнение программы и выполнение тестовых примеров

Ниже представлен результат выполнения тестовых примеров и потенциальных проблем, которые могли сломать код

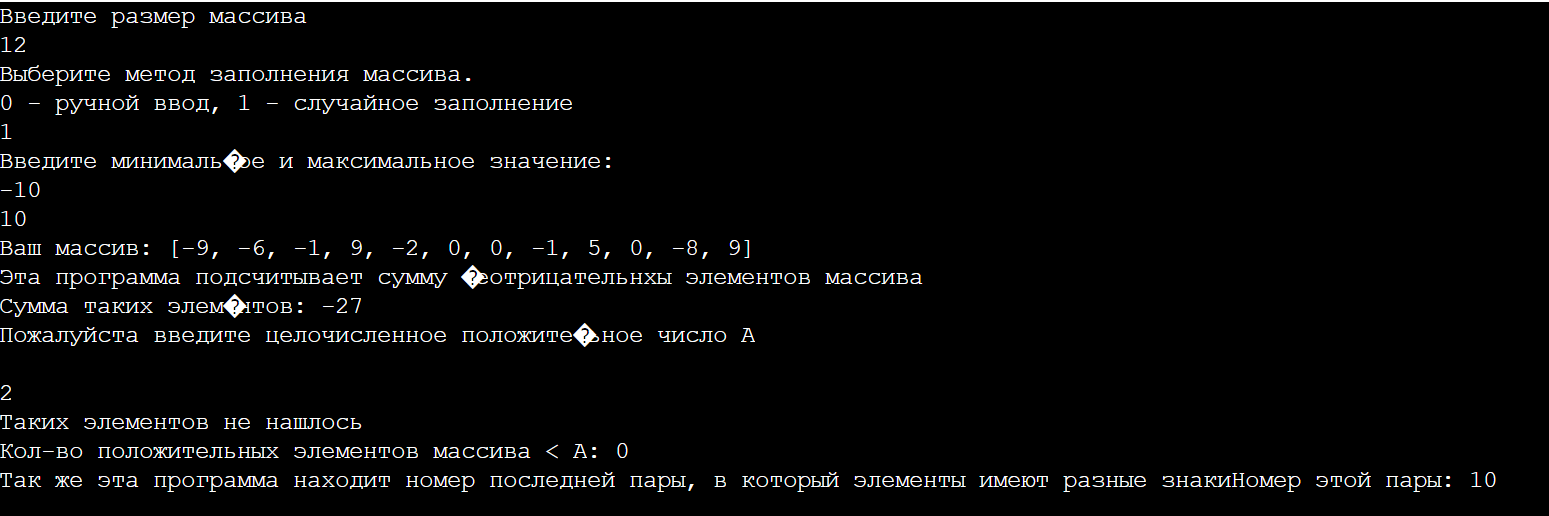


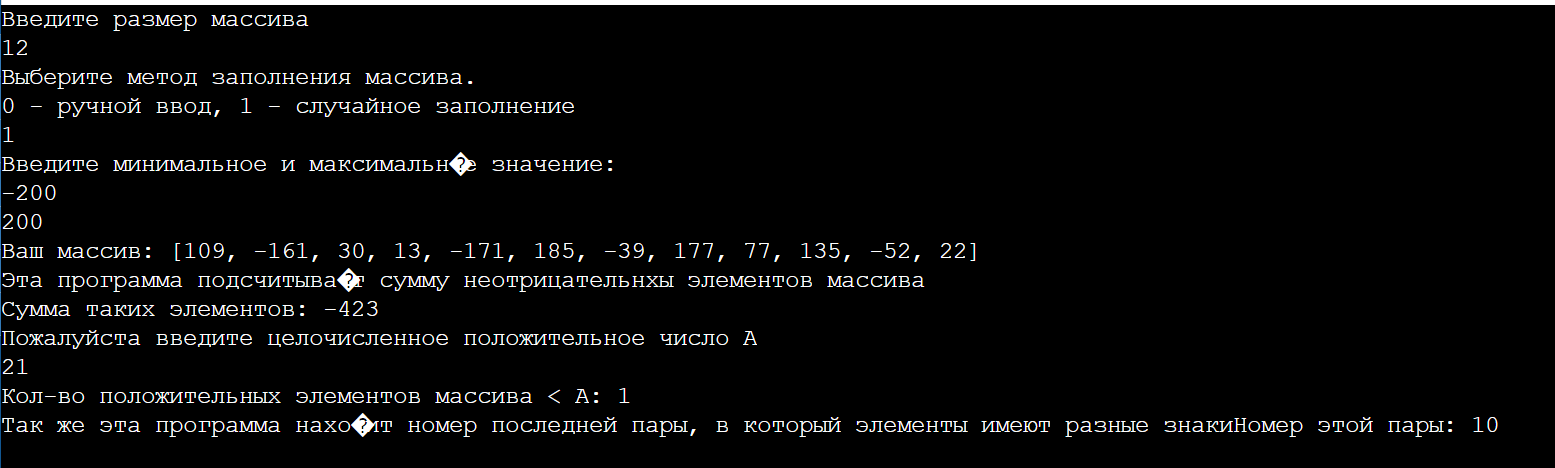
Рисунок  – Первый результат выполнения программы

Рисунок  – Второй результат выполнение программы

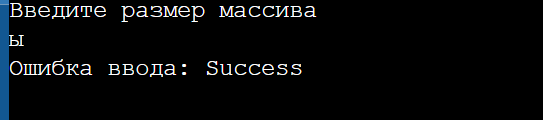


Рисунок – Проверка функции input()

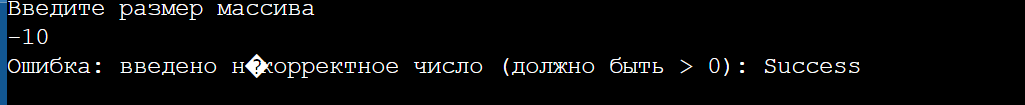


Рисунок  – Проверка функции pozitiv\_input()

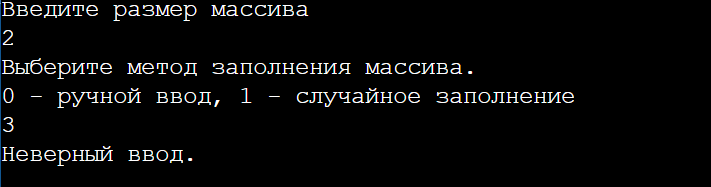


Рисунок  – Проверка неправильного ввода выбора

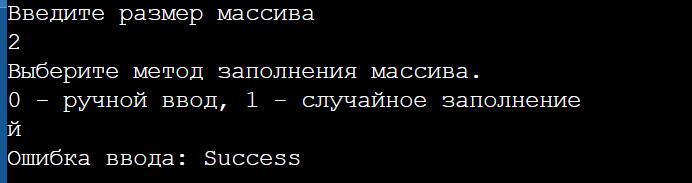


Рисунок  – Еще одна проверка функции input()

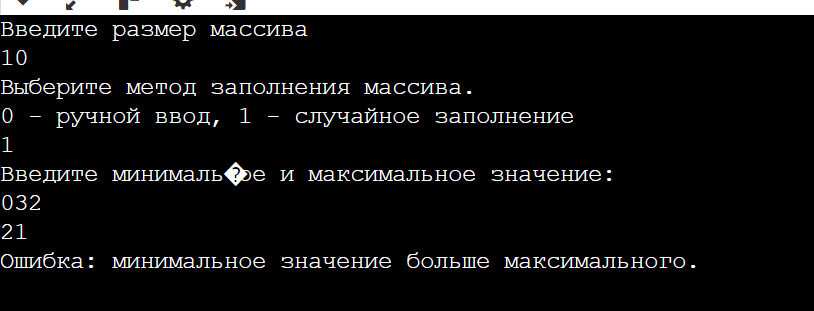


Рисунок – Минимум больше максимума

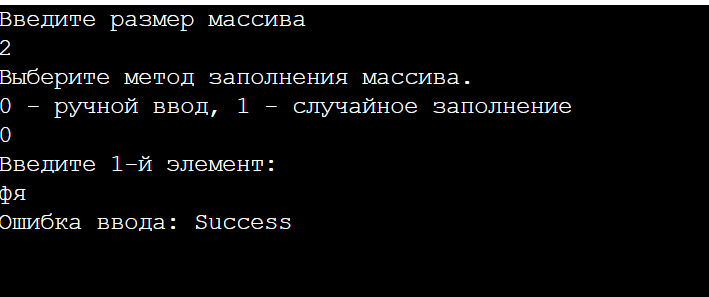


Рисунок  – Еще одна проверка функции input()

5 отметка о выполнение

