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 6

Выполнил: ст. гр. ТКИ-142

Дохолян Герман Оганесович

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

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

Москва 2024

1. Формулировка задания

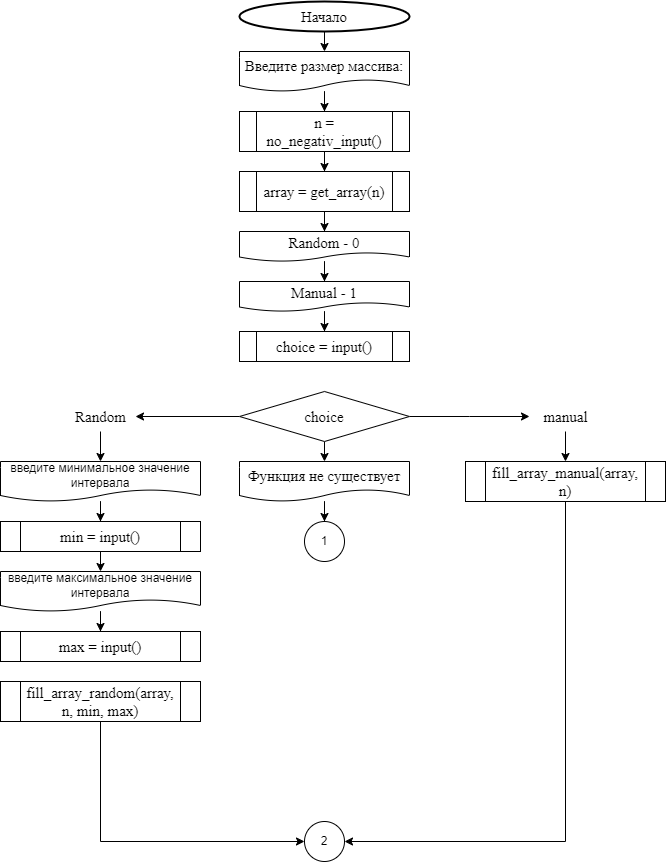
Создать одномерный массив из *n* целых чисел. Реализовать возможность заполнения массива, как случайными числами, так и с помощью клавиатуры по желанию пользователя. Пункты задания организовать в виде функций (методов). Составить блок-схему.

Таблица 1 – Формулировка задания 4.1

|  |  |  |
| --- | --- | --- |
| Вариант | Задачи | Интервал |
| 6 | 1. Найти сумму отрицательных элементов, значения которых кратно 10. 2. Заменить первые k элементов массива на те же элементы в обратном порядке. 3. Определить, есть ли пара соседних элементов с произведением, равным заданному числу. | [-1000;1000] |

1. Блок-схема алгоритма

Блок-схема основного алгоритма представлена ниже (Рисунок 1). Блок-схемы остальных функций представлены ниже (Рисунок 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12).





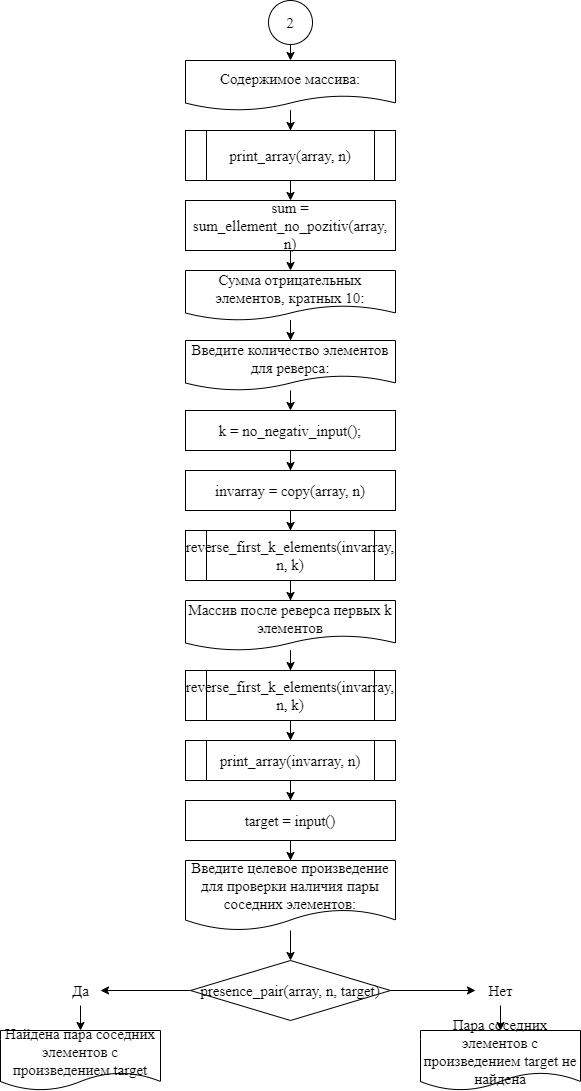


Рисунок  -­ Блок-схема основного алгоритма

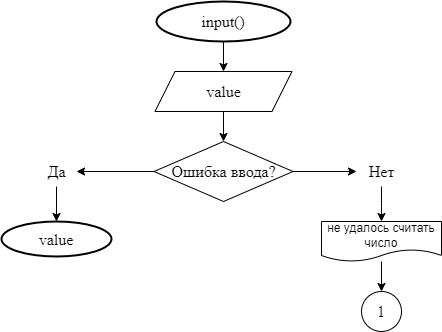


Рисунок 2 – Блок-схема функции input()

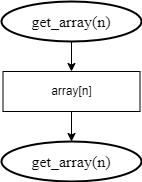


Рисунок 3 – Блок-схема функции get\_array(n)

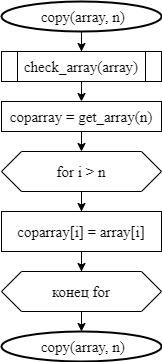


Рисунок 4 – Блок-схема функции copy(array, n)

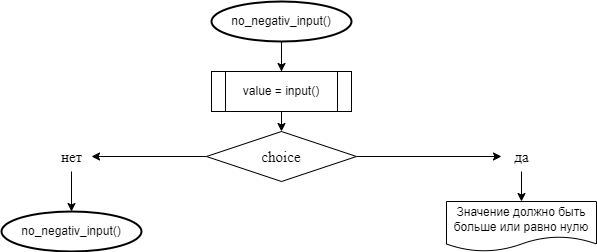


Рисунок 5 – Блок-схема функции no\_negativ\_input()

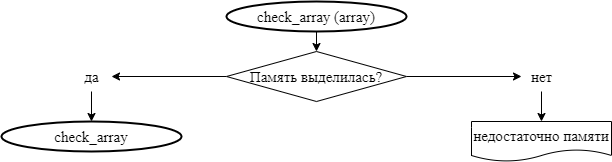


Рисунок 6 – Блок-схема функции check\_array(array)

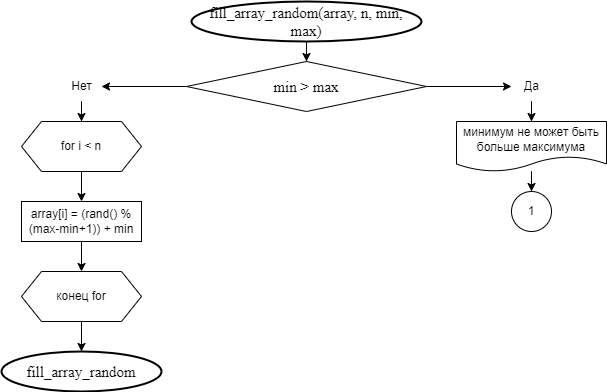


Рисунок 7 – Блок-схема функции fill\_array\_random(array, n, max, min)

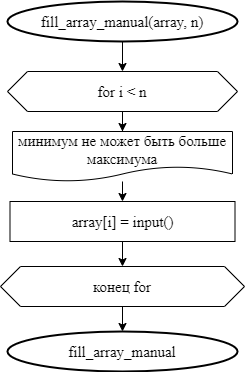


Рисунок 8 – Блок-схема функции fill\_array\_manual (array, n)

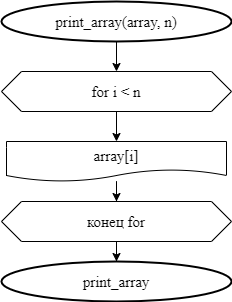


Рисунок 9 – Блок-схема функции print\_array(array, n)

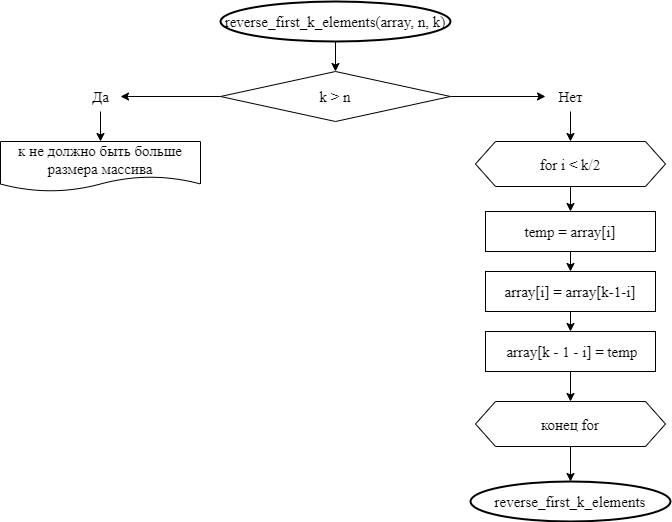


Рисунок 10 – Блок-схема функции reverse\_first\_k\_elements (array, k, n)

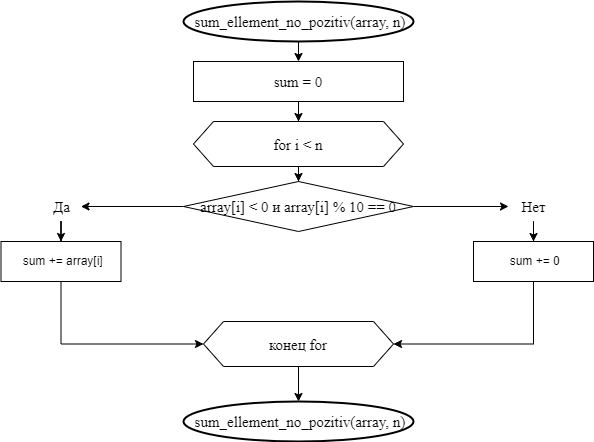
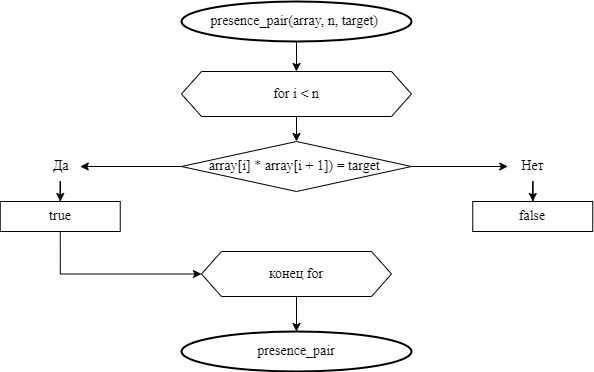
Рисунок 11 – Блок-схема функции sum\_ellement\_no\_pozitiv(array,n)

Рисунок 12 – Блок-схема функции presence\_pair(array, n, target)

1. Текст программы на языке C

#include <stdio.h>

#include <stdlib.h>

#include <errno.h>

#include <time.h>

#include <float.h>

#include <stdbool.h>

/\*\*

\* @brief Считывает введеное значение

\* @return Возвращает введенное значение

\*/

int input(void);

/\*\*

\* @brief Создает массив

\* @param n размер массива

\* @return Возвращает массив с n элементами

\*/

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

/\*\*

\* @brief Проверяет введенное число на неотрицательность

\* @param value вводимое число

\* @return Возвращает ошибку, если число отрицательно или равно 0

\*/

int no\_negativ\_input(void);

/\*\*

\* @brief Проверяет массив

\* @param array массив

\* @return возвращает ошибку, в случае недостатка памяти для массива

\*/

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

/\*\*

\* @brief Копирует исходный массив

\* @param array исходный массив

\* @param n число элементов массива

\* @return Возвращает скопированный массив

\*/

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

/\*\*

\* @brief Заполняет массив случайными числами

\* @param array массив

\* @param n количество элементов в массиве

\* @param min начало диапазона

\* @param max конец диапазона

\* @return возвращает массив, заполненный случайными числами

\*/

void fill\_array\_random(int\* array, const size\_t n, const int min, const int max);

/\*\*

\* @brief Заполнение массива при помощи ввода элементов с клавиатуры

\* @param array массив

\* @param n количество элементов в массиве

\* @return возвращает массив, заполненный числами, введеными вручную

\*/

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

/\*\*

\* @brief Выводит исходный массив

\* @param array массив

\* @param n количество элементов в массиве

\* @return Возвращает вывод исходного массива

\*/

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

/\*\*

\* @brief Находит сумму отрицательных элементов массива, кратных 10

\* @param array Указатель на массив, который будет проверяться

\* @param n Количество элементов в массиве

\* @return Возвращает сумму отрицательных элементов, кратных 10

\*/

int sum\_ellement\_no\_pozitiv (const int\* array, const size\_t n);

/\*\*

\* @brief Заменяет первые k элементов массива на те же элементы в обратном порядке

\* @param array Указатель на массив, который будет изменен

\* @param n Количество элементов в массиве

\* @param k Количество элементов для замены

\*/

void reverse\_first\_k\_elements(int\* array, size\_t n, size\_t k);

/\*\*

\* @brief Проверка, есть ли пара соседних элементов с заданным произведнием

\* @return true, если есть пара; false, если нет

\*/

bool presence\_pair(int\* array, size\_t n, int target);

/\*\*

\* @param random заполнение массива случайными числами

\* @param manual заполнение массива пользователем

\*/

enum request

{

Random, manual

};

/\*\*

\* @brief Точка входа в программу

\* @return Возвращает 0 в случае успеха

\*/

int main(void)

{

printf("Введите размер массива: ");

size\_t n = no\_negativ\_input();

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

printf("Random - %d\n", Random);

printf("Manual - %d\n", manual);

int choice = input();

switch (choice)

{

case Random:

srand(time(0)); // Инициализация генератора случайных чисел

printf("введите минимальное значение интервала \n");

const int min = input();

printf("введите максимальное значение интервала \n");

const int max = input();

fill\_array\_random(array, n, min, max);

break;

case manual:

fill\_array\_manual(array, n);

break;

default:

errno = ERANGE;

perror("Function does not exist\n");

exit(EXIT\_FAILURE);

break;

}

printf("Содержимое массива: ");

print\_array(array, n);

int sum = sum\_ellement\_no\_pozitiv(array, n);

printf("Сумма отрицательных элементов, кратных 10: %d\n", sum);

printf("Введите количество элементов для реверса: ");

size\_t k = no\_negativ\_input();

int\* invarray = copy(array, n);

reverse\_first\_k\_elements(invarray, n, k);

printf("Массив после реверса первых %zu элементов: ", k);

print\_array(invarray, n);

free(invarray);

int target = input();

printf("Введите целевое произведение для проверки наличия пары соседних элементов: ");

if (presence\_pair(array, n, target)) {

printf("Найдена пара соседних элементов с произведением %d.\n", target);

} else {

printf("Пара соседних элементов с произведением %d не найдена.\n", target);

}

free(array);

return 0;

}

int input(void)

{

int value = 0;

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

if (result != 1)

{

errno = EIO;

perror("ошибка ввода");

exit(EXIT\_FAILURE);

}

return value;

}

int\* get\_array(const size\_t n)

{

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

return array;

}

int no\_negativ\_input(void)

{

int value = input();

if (value <= 0)

{

errno = EINVAL;

perror("Значение должно быть больше или равно нулю");

exit(EXIT\_FAILURE);

}

return value;

}

void check\_array(const int\* array)

{

if (array == NULL)

{

errno = ENOMEM;

perror("недостаточно памяти");

exit(EXIT\_FAILURE);

}

}

void fill\_array\_random(int\* array, const size\_t n, const int min, const int max)

{

check\_array(array);

if (min > max)

{

errno = EINVAL;

perror("минимум не может быть больше максимума");

exit(EXIT\_FAILURE);

}

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

{

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

}

}

void fill\_array\_manual(int\* array, const size\_t n)

{

check\_array(array);

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

{

printf("введите элемент %zu: ", i);

array[i] = input();

}

}

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

{

check\_array(array);

int\* coparray = get\_array(n);

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

{

coparray[i] = array[i];

}

return coparray;

}

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

{

check\_array(array);

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

{

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

}

printf("\n");

}

int sum\_ellement\_no\_pozitiv (const int\* array, const size\_t n)

{

int sum = 0;

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

{

if (array[i] < 0 && array[i] % 10 == 0)

{

sum += array[i];

}

}

return sum;

}

void reverse\_first\_k\_elements(int\* array, size\_t n, size\_t k)

{

if (k > n)

{

errno = EINVAL;

perror("к не должно быть больше размера массива");

exit(EXIT\_FAILURE);

}

for (size\_t i = 0; i < k / 2; i++)

{

int temp = array[i];

array[i] = array[k - 1 - i];

array[k - 1 - i] = temp;

}

}

bool presence\_pair(int\* array, size\_t n, int target)

{

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

{

if (array[i] \* array[i + 1]) - target < DBL\_EPSILON;

{

return true;

}

}

return false;

}

1. Результаты выполнения программы

Результаты выполнения программы представлены ниже (Рисунок 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30)

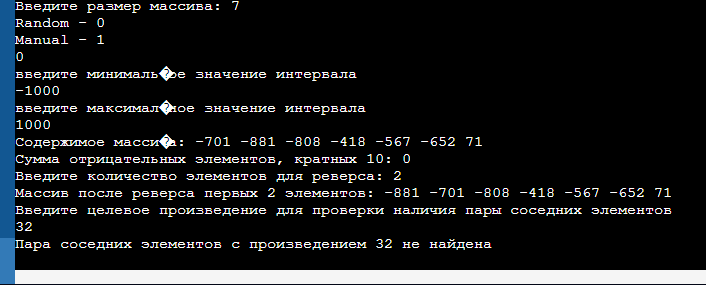


Рисунок 14 – Результаты выполнения программы

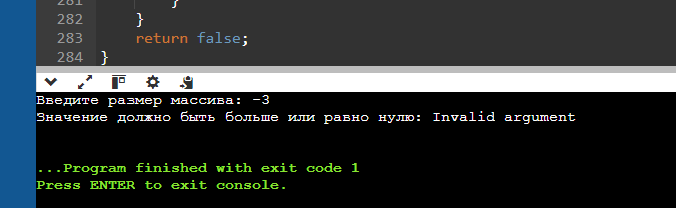


Рисунок 15 – Результаты выполнения программы при вводе отрицательного числа в величину размера массива.

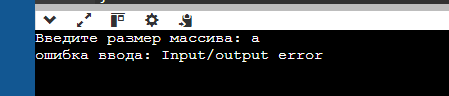


Рисунок 16 – Результаты выполнения программы если размер массива – буква

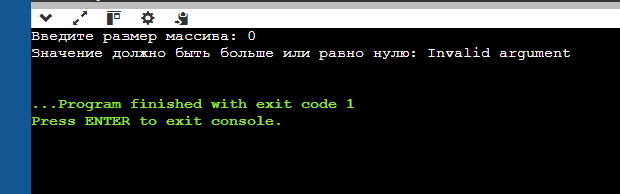


Рисунок 17 – Результаты выполнения программы если размер массива – нуль

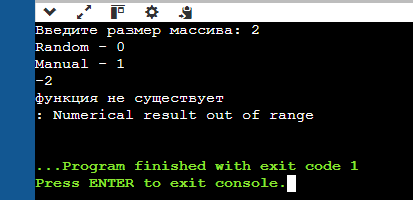


Рисунок 18 - Результаты выполнения программы, если число выбора отрицательно

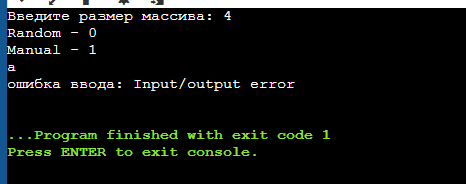


Рисунок 19 – Результаты выполнения программы, если введенный выбор – буква

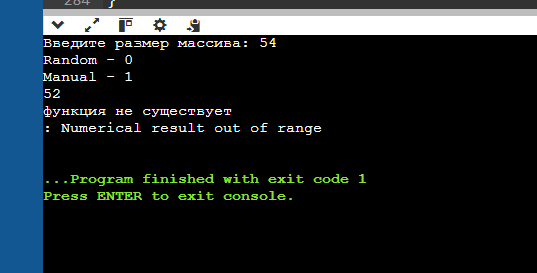
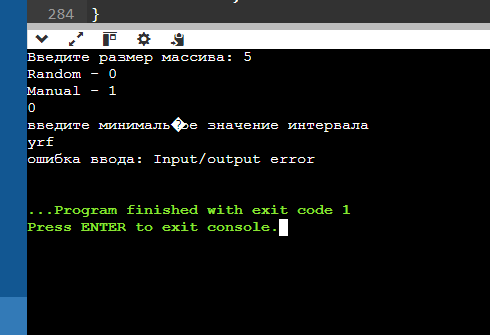


Рисунок 20 – Результат выполнения программы, если выбор больше диапазона

  
Рисунок 21 – Результат выполнения программы, если минимальное значения интервала – слово

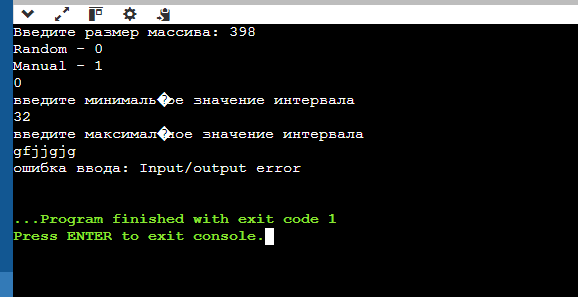


Рисунок 22 – Результат выполнения программы, если введенный максимум интервала – слово

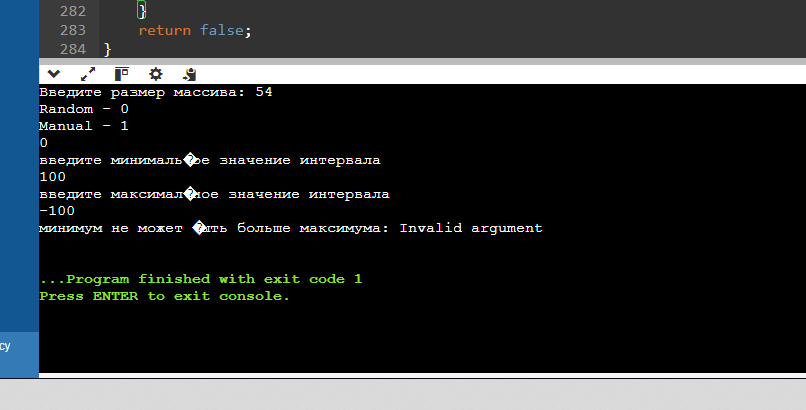


Рисунок 23 – Результат выполнения программы, если минимум интервала больше максимума

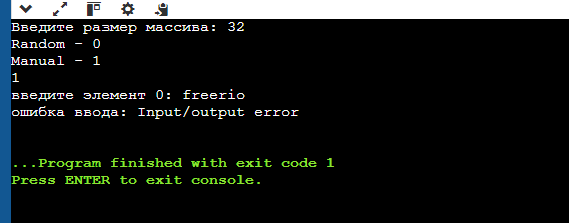


Рисунок 24 – Результат выполнения программы, если введенный элемент массива – слово

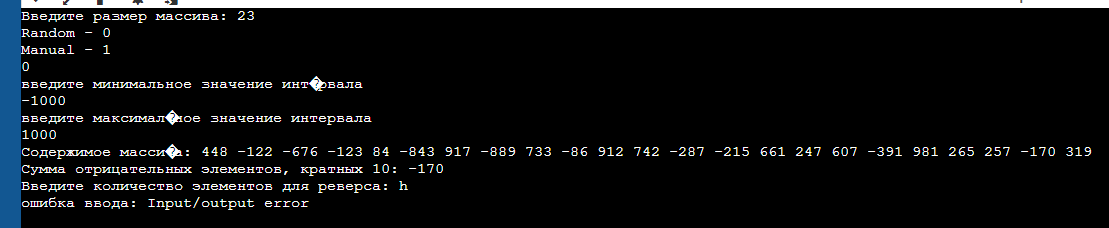


Рисунок 25 – Результат выполнения программы, если введенное значение k – буква

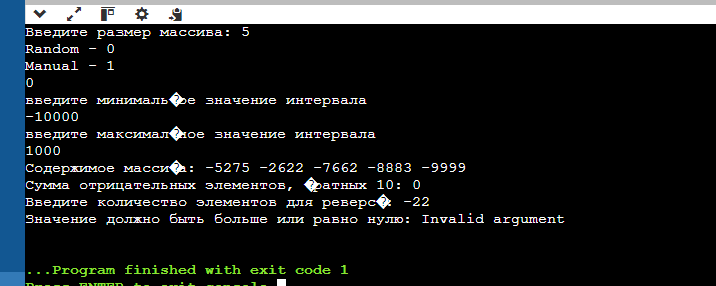


Рисунок 26 – Результат выполнения программы, если введенное значение k отрицательно

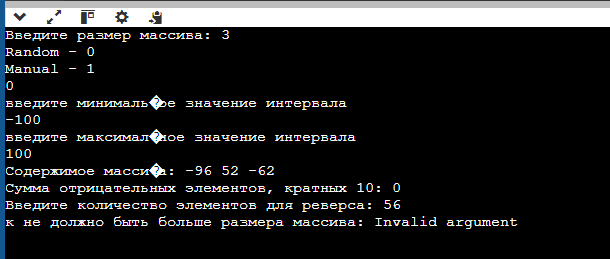


Рисунок 27 – Результат выполнения программы, если введенное значение k больше n

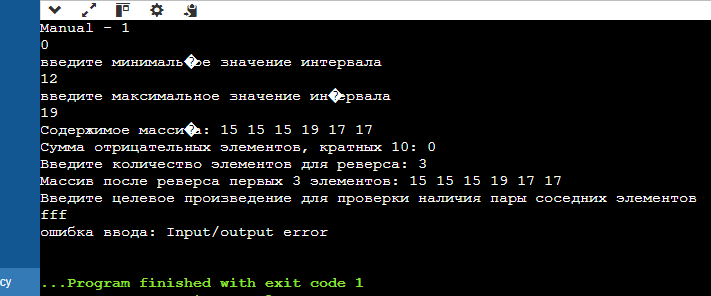


Рисунок 28 – Результат выполнения программы, если введенное значение искомого произведения – буква

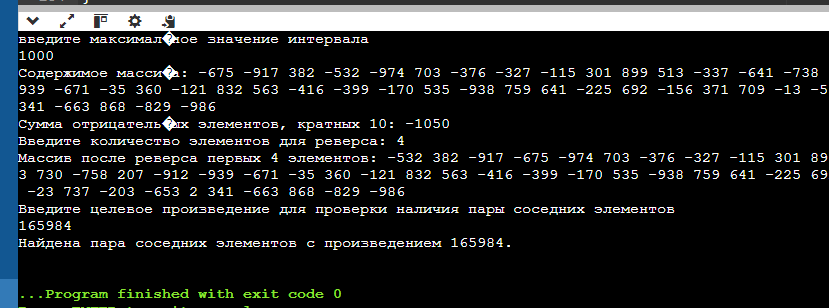


Рисунок 29 – Результат выполнения программы, когда искомое произведение найдено

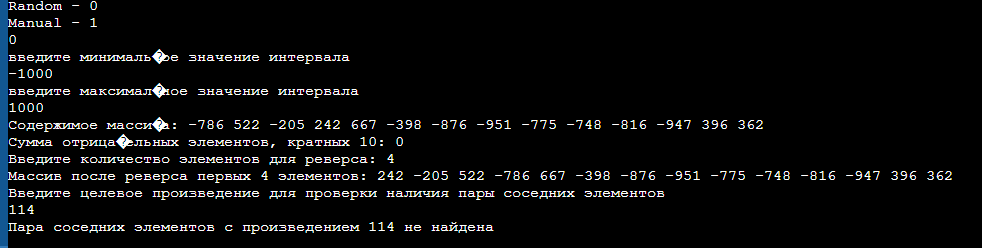


Рисунок 30 – Результат выполнения программы, если искомое произведение не найдено

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

