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 №1

По дисциплине «Языки программирования»

ВАРИАНТ 11

Выполнил: ст. гр. ТКИ-141

Колчихина Виктория Андреевна

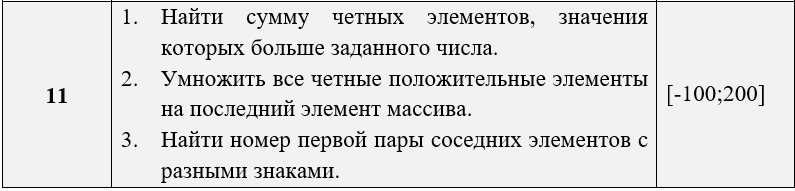
(Проверил: к.т.н, доц. Балакина Е. П.)

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  – Исходные данные



1. Блок-схема алгоритма

Блок-схема основного алгоритма представлена ниже (Рисунок 1). Блок-схемы функций для проверки чисел PrintArray(array, arrayLength) и dInput(void) представлены ниже (Рисунок 2 ). Блок-схема используемой функции sizetInput(void) представлена ниже (Рисунок 3). Блок-схемы функций для проверки указателя и выделения памяти массива  VerifyPointer(const int\* pointer) и InitializeArray(const size\_t arrayLength) представлены ниже (Рисунок 4). Блок-схемы функций для вывода конечного результата Task1(array, arrayLength, k),  Task2(array, arrayLength), Task3(array, arrayLength) представлены ниже (Рисунок 5). Блок-схемы функций для копирования и вывода элементов массива  CopyArray(const int\* arrayToCopy, const size\_t arrayLength) и PrintArray(const int\* array, const size\_t arrayLength) представлены ниже (Рисунок 6). Блок-схемы функций FillArrayManually и FillArrayRandomly представлены ниже (Рисунок 7)

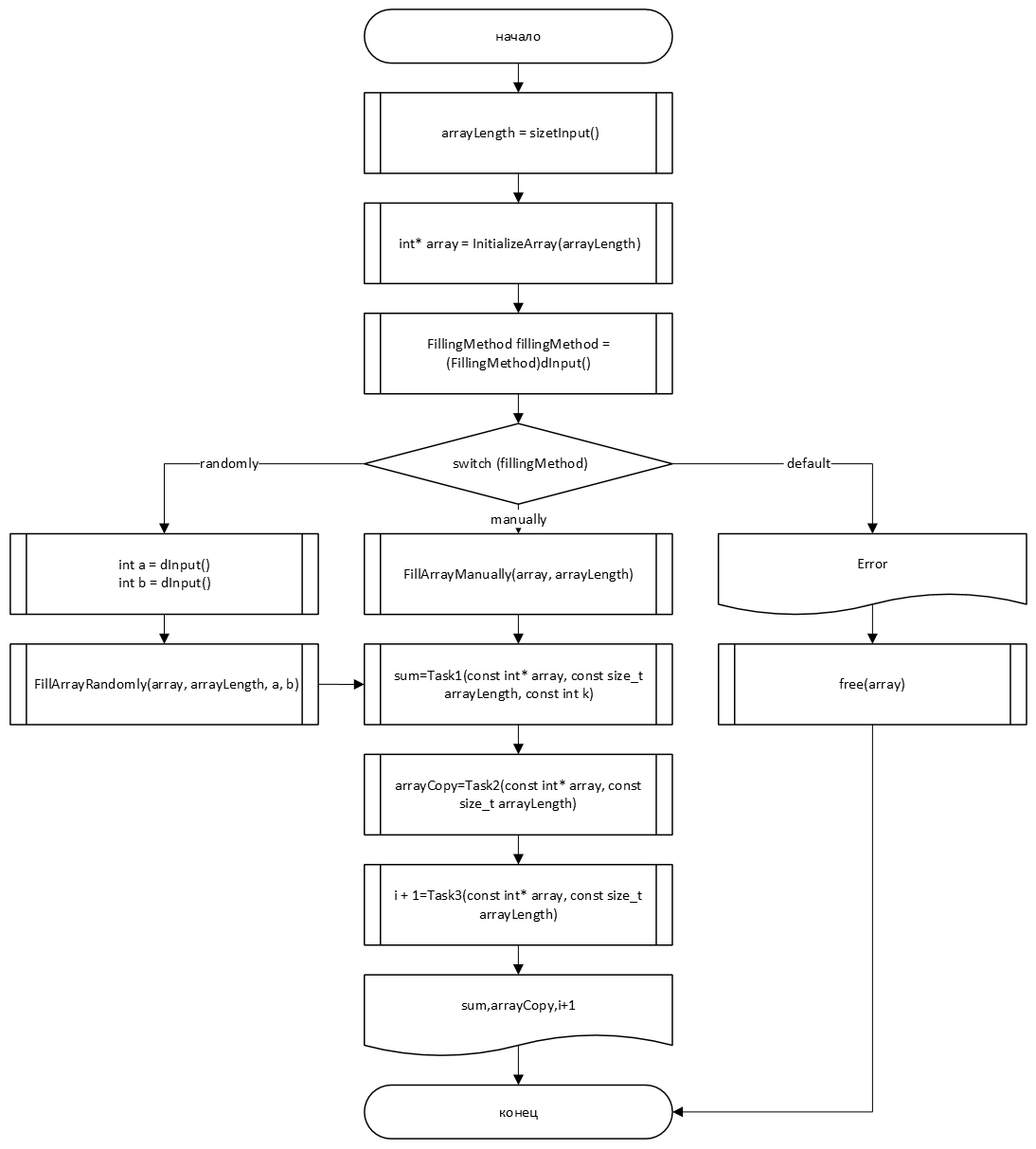


Рисунок  ­ Блок-схема основного алгоритма

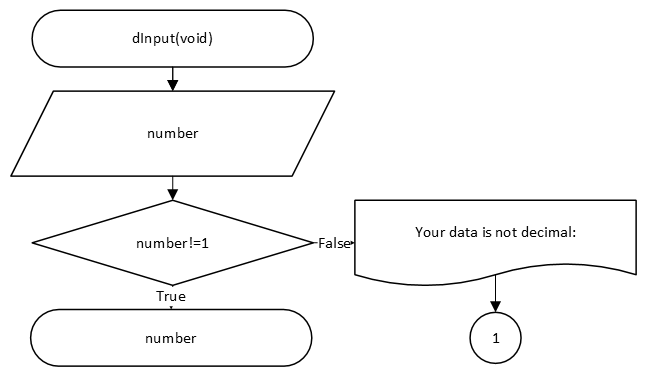
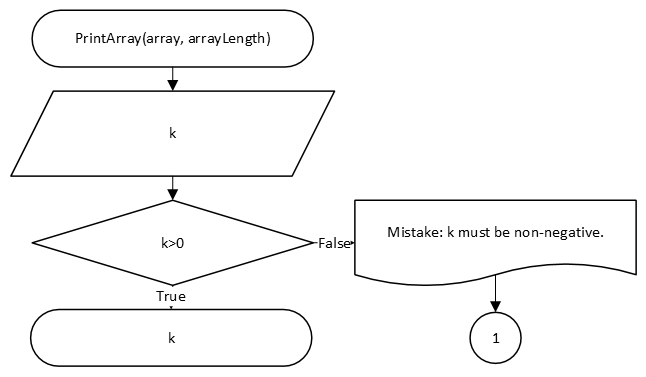


Рисунок  – Блок-схемы используемых функций проверки чисел(PrintArray(array, arrayLength) и dInput(void))

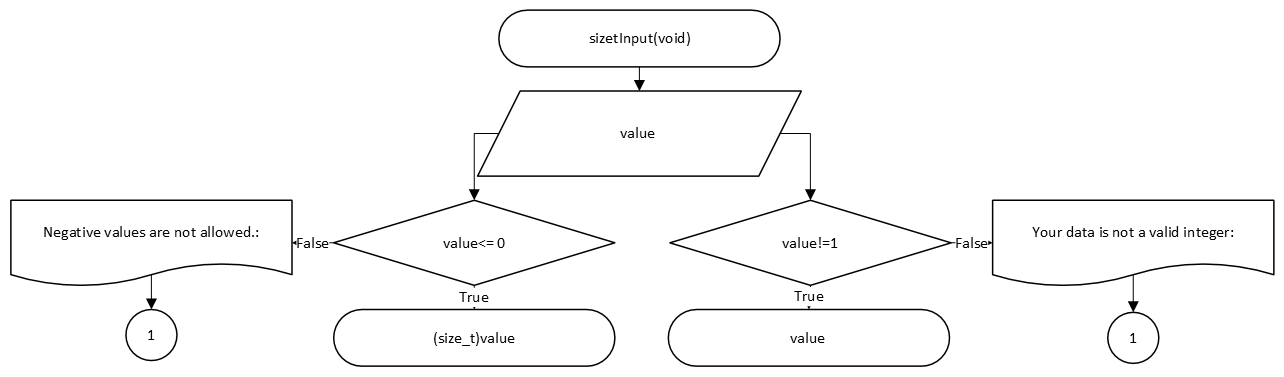


Рисунок 3 – Блок-схема используемой функции (sizetInput(void))

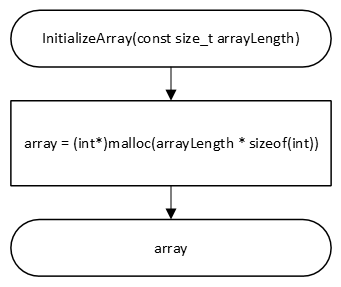
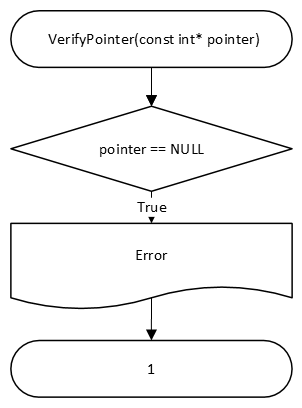


Рисунок 4 –  Блок-схемы используемых функций для проверки указателя и выделения памяти массива ( VerifyPointer(const int\* pointer) и InitializeArray(const size\_t arrayLength))

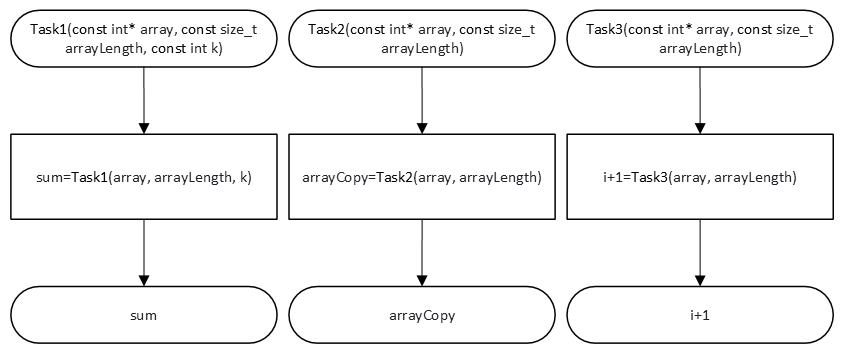


Рисунок 5 – Блок-схемы используемых функций для вывода конечных результатов (Task1(array, arrayLength, k),  Task2(array, arrayLength), Task3(array, arrayLength))

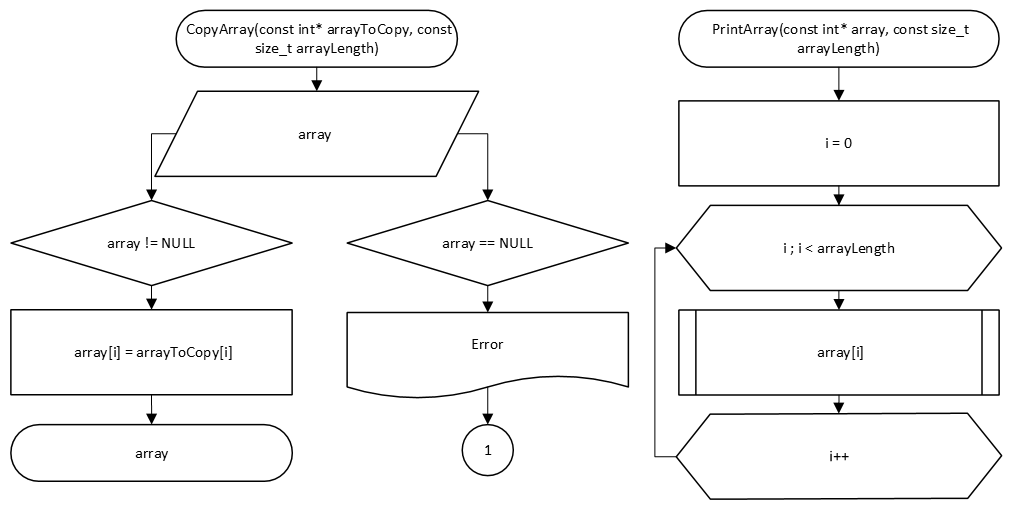


Рисунок 6 – Блок-схемы используемых функций для копирования и вывода элементов массива ( CopyArray(const int\* arrayToCopy, const size\_t arrayLength) и PrintArray(const int\* array, const size\_t arrayLength))

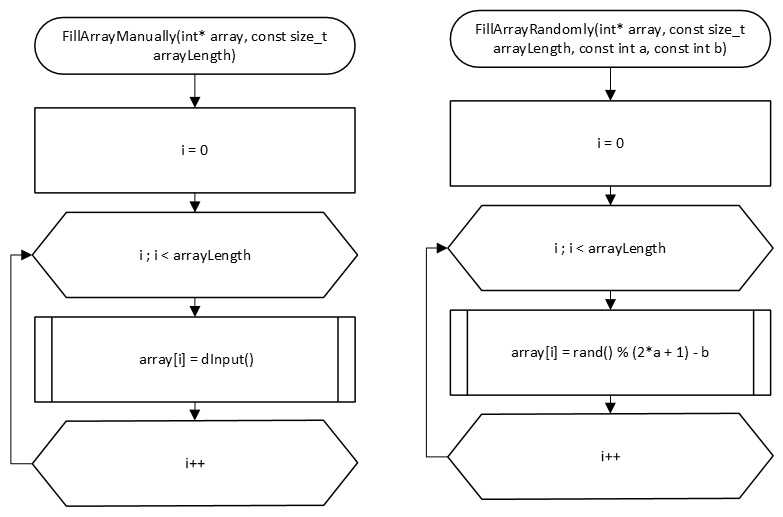


Рисунок 7 - Блок-схемы используемых функций FillArrayManually и FillArrayRandomly

1. Текст программы на языке C

#include <stdio.h>

#include <stdlib.h>

#include <time.h>

#include <math.h>

/\*\*

\* @brief считывает целое число.

\* @return возращает целое число.

\*/

int dInput(void);

/\*\*

\* @brief считывает значение size\_t.

\* @return значение size\_t.

\*/

size\_t sizetInput(void);

/\*\*

\* @brief проверяет что указатель не равен NULL.

\* @param pointer указатель для проверки.

\*/

void VerifyPointer(const int\* pointer);

/\*\*

\* @brief выделяет память для массива целых чисел.

\* @param arrayLength длина массива.

\* @return указатель на выделенный массив.

\*/

int\* InitializeArray(const size\_t arrayLength);

/\*\*

\* @brief копирует массив целых чисел.

\* @param arrayToCopy указатель на массив целых чисел который необходимо скопировать.

\* @param arrayLength длина массива которую нужно скопировать.

\* @return указатель на вновь выделенный массив содержащий скопированные значения.

\*/

int\* CopyArray(const int\* arrayToCopy, const size\_t arrayLength);

/\*\*

\* @brief выводит элементы массива.

\* @param array указатель на массив.

\* @param arrayLength длина массива.

\*/

void PrintArray(const int\* array, const size\_t arrayLength);

/\*\*

\* @brief заполняет массив данными введеными пользователем.

\* @param array указатель на массив.

\* @param arrayLength длина массива.

\*/

void FillArrayManually(int\* array, const size\_t arrayLength);

/\*\*

\* @brief заполняет массив случайными значениями.

\* @param array указатель на массив.

\* @param arrayLength длина массива.

\* @param a начало диапазона.

\* @param b конец диапазона.

\*/

void FillArrayRandomly(int\* array, const size\_t arrayLength, const int a, const int b);

/\*\*

\* @brief выполняет операцию над целочисленным массивом.

\* @param array указатель на входной целочисленный массив.

\* @param arrayLength длина входного массива.

\*/

int Task1(const int\* array, const size\_t arrayLength, const int k);

/\*\*

\* @brief создает новый целочисленный массив на основе входного массива.

\* @param array указатель на входной целочисленный массив.

\* @param arrayLength длина входного массива.

\*/

int\* Task2(const int\* array, const size\_t arrayLength);

/\*\*

\* @brief подсчитывает что-либо в целочисленном массиве.

\* @param array указатель на входной целочисленный массив.

\* @param arrayLength длина входного массива.

\*/

size\_t Task3(const int\* array, const size\_t arrayLength);

/\*\*

\* @brief перечисление для методов заполнения массива.

\*

\* это перечисление определяет два метода заполнения массива:

\* - вручную: массив заполняется данными введенными пользователем.

\* - случайным образом: массив заполняется случайными значениями введенными пользователем.

\*/

typedef enum

{

manually = 1,

randomly = 2

} FillingMethod;

/\*\*

\* @brief точка входа в программу

\* @return возращает ноль в случае успеха

\*/

int main(void)

{

srand(time(NULL));

puts("Введите длину массива:");

const size\_t arrayLength = sizetInput();

int\* array = InitializeArray(arrayLength);

puts("Введите способ заполнения массива:");

const FillingMethod fillingMethod = (FillingMethod)dInput();

switch (fillingMethod)

{

case manually:

FillArrayManually(array, arrayLength);

break;

case randomly:

puts("Введите диапазон генерации чисел [a, b]");

const int a = dInput();

const int b = dInput();

FillArrayRandomly(array, arrayLength, a, b);

break;

default:

puts("Неверное значение для метода заполнения:");

free(array);

return 1;

}

PrintArray(array, arrayLength);

{

puts("Введите значение k:");

const int k = dInput();

if (k < 0)

{

puts("Ошибка: k должно быть неотрицательным.");

free(array);

return 1;

}

}

puts("1.");

int k;

printf("Сумма = %d", Task1(array, arrayLength, k));

int\* secondTaskArray = Task2(array, arrayLength);

puts("2.");

PrintArray(secondTaskArray, arrayLength);

puts("3.");

printf("Номер первой пары = %zu", Task3(array, arrayLength));

free(array);

free(secondTaskArray);

return 0;

}

int dInput(void)

{

int number = 0;

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

{

puts("Ваши данные не являются десятичными:");

exit(EXIT\_FAILURE);

}

return number;

}

size\_t sizetInput(void)

{

int value = dInput();

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

{

puts("Ваши данные не являются допустимым целым числом:");

exit(EXIT\_FAILURE);

}

if (value <= 0)

{

puts("Отрицательные значения не допускаются:");

exit(EXIT\_FAILURE);

}

return (size\_t)value;

}

void VerifyPointer(const int\* pointer)

{

if (pointer == NULL)

{

puts("Ошибка выделения памяти:");

exit(EXIT\_FAILURE);

}

}

int\* InitializeArray(const size\_t arrayLength)

{

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

VerifyPointer(array);

return array;

}

int\* CopyArray(const int\* arrayToCopy, const size\_t arrayLength)

{

VerifyPointer(arrayToCopy);

int\* array = InitializeArray(arrayLength);

if (array == NULL) {

return NULL;

}

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

{

array[i] = arrayToCopy[i];

}

return array;

}

void PrintArray(const int\* array, const size\_t arrayLength)

{

VerifyPointer(array);

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

{

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

}

printf("\n");

}

void FillArrayManually(int\* array, const size\_t arrayLength)

{

VerifyPointer(array);

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

{

array[i] = dInput();

}

}

void FillArrayRandomly(int\* array, const size\_t arrayLength, const int a, const int b)

{

VerifyPointer(array);

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

{

array[i] = rand() % (2\*a + 1) - b; // [-100; 200]

}

}

int Task1(const int\* array, const size\_t arrayLength, const int k)

{

VerifyPointer(array);

int sum = 0;

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

{

if ((array[i] % 2 == 0) && (array[i] > k))

{

sum += array[i];

}

}

return sum;

}

int\* Task2(const int\* array, const size\_t arrayLength)

{

VerifyPointer(array);

int\* arrayCopy = CopyArray(array, arrayLength);

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

{

if ((array[i] % 2 == 0) && (array[i] > 0))

{

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

}

}

return arrayCopy;

}

size\_t Task3(const int\* array, const size\_t arrayLength)

{

VerifyPointer(array);

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

{

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

{

return (i + 1);

}

}

puts("Такой пары нет!");

return 0; // в случае если такой пары нет

}

1. Результаты выполнения программы

Результаты выполнения программы представлены ниже (Рисунок 6).

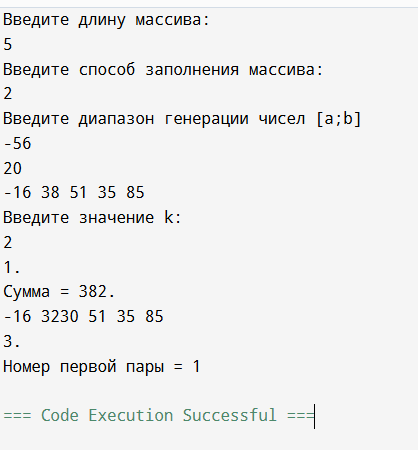


Рисунок 8 – Результаты выполнения программы

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

