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

Горковец Анна Сергеевна

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

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

Москва 2023

Оглавление

[1. Задание 4.1 3](#_Toc155803276)

[1.1. Формулировка задания 3](#_Toc155803277)

[1.2. Блок-схема алгоритма 4](#_Toc155803278)

[1.3. Код задания 4.1 15](#_Toc155803279)

[1.4. Решение тестового примера 19](#_Toc155803280)

[1.5. Зачёт задания в GitHub 20](#_Toc155803281)

[2. Задание 4.2 21](#_Toc155803282)

[2.1. Формулировка задания 21](#_Toc155803283)

[2.2. Блок-схема алгоритма 22](#_Toc155803284)

[2.3. Решение тестового примера 38](#_Toc155803285)

[2.4. Зачёт задания в GitHub 41](#_Toc155803286)

[3. Задание 4.3 41](#_Toc155803287)

[3.1. Формулировка задания 41](#_Toc155803288)

[3.2. Блок-схема алгоритма 42](#_Toc155803289)

[3.3. Код задания 4.3 50](#_Toc155803290)

[3.4. Решение тестового примера 57](#_Toc155803291)

[3.5. Зачёт задания в GitHub 58](#_Toc155803292)

# 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 – Формулировка задания 3.1

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

* 1. Блок-схема алгоритма

Блок-схема алгоритма представлена на рисунках (Рисунки 1-12)

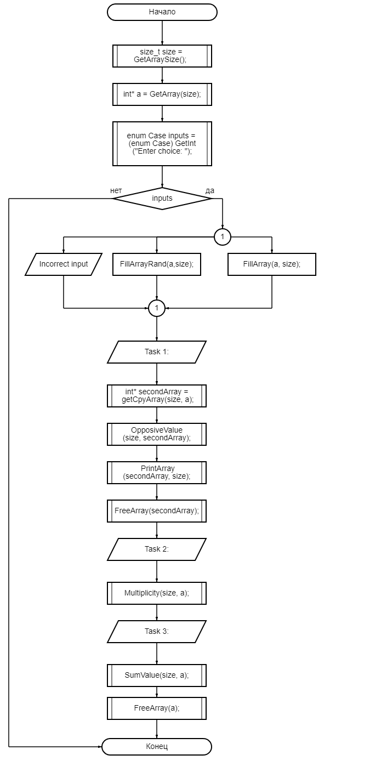


Рисунок 1 – Блок-схема функции main()

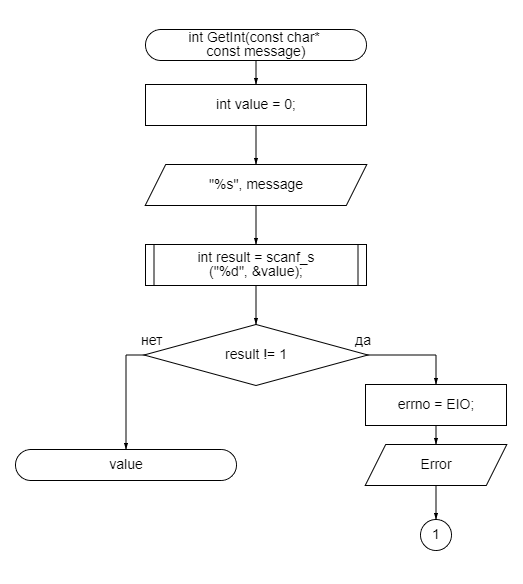


Рисунок 2 – Блок-схема функции GetInt ()

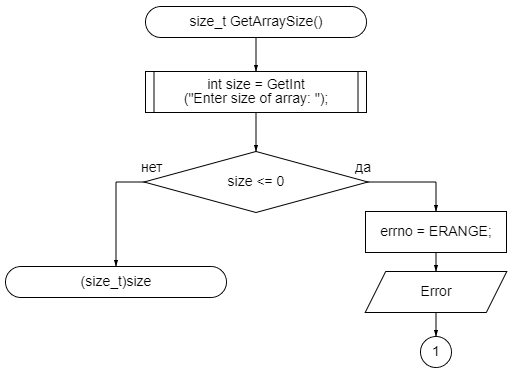


Рисунок 3 – Блок-схема к функции GetArraySize()

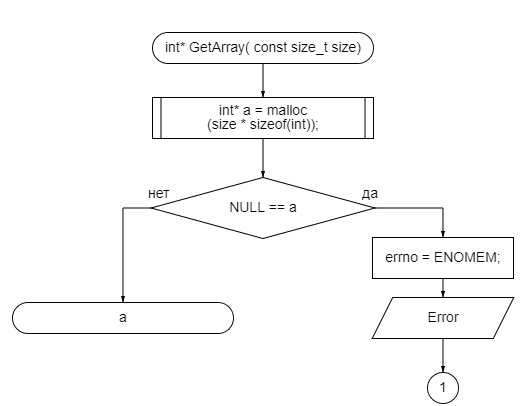


Рисунок 4 – Блок-схема функции GetArray ()

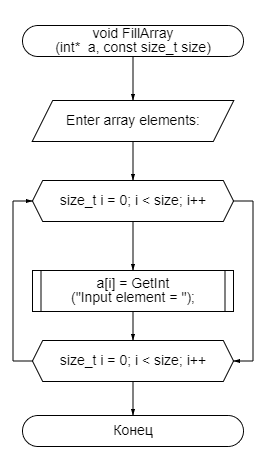


Рисунок *5* – Блок-схема функции FillArray ()

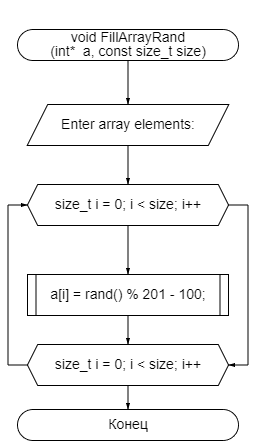


Рисунок *6*– Блок-схема функции FillArrayRand ()

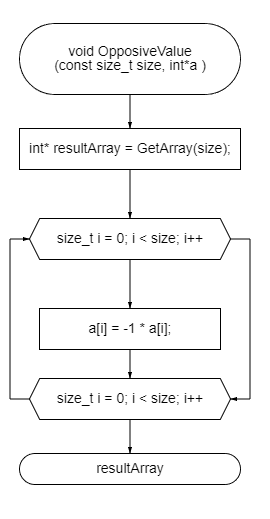


Рисунок 7– Блок-схема функции OpposiveValue ()

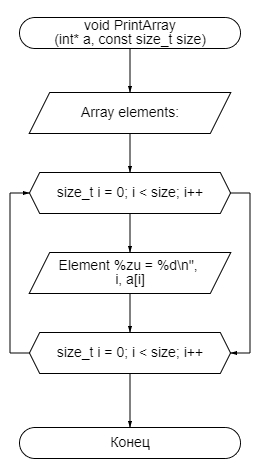


Рисунок 8– Блок-схема функции PrintArray ()

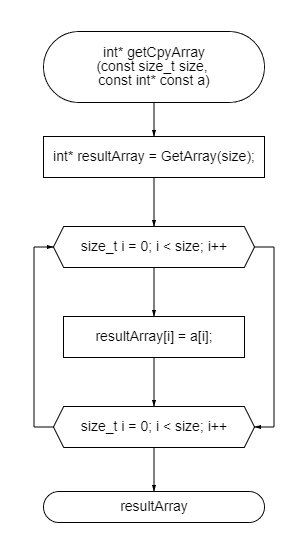


Рисунок 9– Блок-схема функции getCpyArray ()

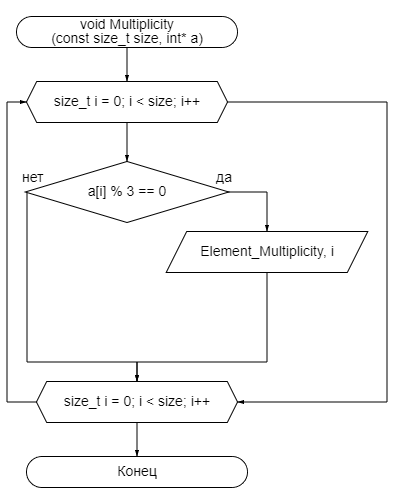


Рисунок 10– Блок-схема функции Multiplicity ()

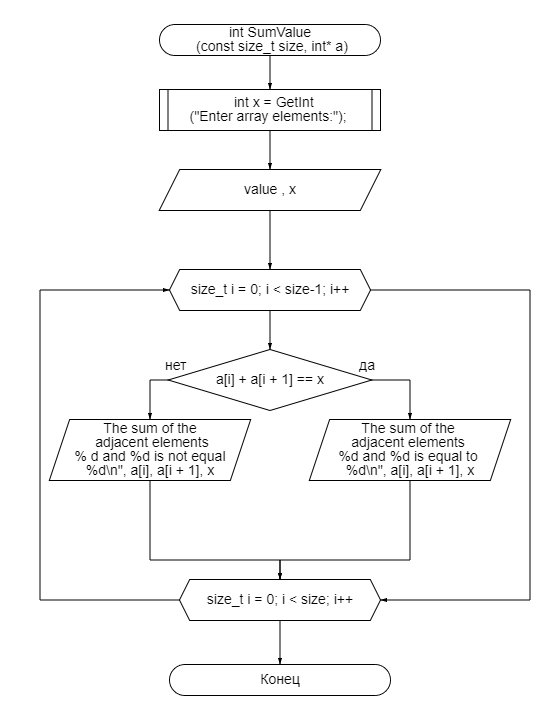


Рисунок 11– Блок-схема функции SumValue ()

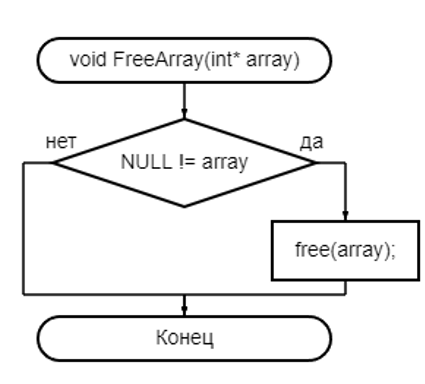


Рисунок 12– Блок-схема функции FreeArray ()

* 1. Код задания 4.1

#include <stdio.h>

#include <malloc.h>

#include <errno.h>

#include <stdlib.h>

/\*\*

\* @brief считывает и проверяет ввод переменной типа int

\* @return Возвращает считанное значение

\*/

int GetInt(const char\* const message);

/\*\*

\* @brief функция вывод значения элементов массива с противоположным знаком

\* @param size - размер массива

\* @param a - массив

\*/

void OpposiveValue(const size\_t size, int\* a);

/\*\*

\* @brief выводит индексы тех элементов, значения которых кратны 3

\* @param size - размер массива

\* @param a - массив

\*/

void Multiplicity(const size\_t size, int\* a);

/\*\*

\* @brief фунция определяет, есть ли пара соседних элементов с суммой, равной заданному числу.

\* @param x- число для сравнения пары соседних элементов.

\* @param size - размер массива

\* @param a - массив

\*/

int SumValue(const size\_t size, int\* a);

/\*\*

\* @brief функция печатает массив

\* @param size - размер массива

\* @param a - массив

\*/

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

/\*\*

\* @brief функция для создания массива

\* @param а - название массива

\* @param size - размер массива

\* @return возвращает массив

\*/

int\* GetArray(const size\_t size);

/\*\*

\* @brief функция для заполнения массива в ручную

\* @param size - размер массива

\* @param a - массив

\*/

void FillArray(int\* a, const size\_t size);

/\*\*

\* @brief функция для заполнения массива в рандомными числами

\* @param size - размер массива

\* @param a - массив

\*/

void FillArrayRand(int\* a, const size\_t size);

/\*\*

\* @brief функция очищает массив

\* @param a - массив

\*/

void FreeArray(int\* a);

/\*\*

\* @brief функция копирует массив

\* @param resultArray - копия массива

\* @param size - размер массива

\* @param a - массив

\*/

int\* getCpyArray(const size\_t size, const int\* const a);

/\*\*

\* @brief функция для определения размера массива

\* @return возвращает размер массива

\*/

size\_t GetArraySize();

enum Case

{

random = 1,

enter = 2,

};

int main()

{

size\_t size = GetArraySize();

int\* a = GetArray(size);

enum Case inputs = (enum Case) GetInt("Enter choice: ");

switch (inputs)

{

case random:

FillArrayRand(a,size);

break;

case enter:

FillArray(a, size);

break;

default:

puts("Incorrect input.\n");

return 1;

break;

}

puts("Task 1:");

int\* secondArray = getCpyArray(size, a);

OpposiveValue(size, secondArray);

PrintArray(secondArray, size);

FreeArray(secondArray);

puts("Task 2:");

Multiplicity(size, a);

puts("Task 3:");

SumValue(size, a);

FreeArray(a);

return 0;

}

int GetInt(const char\* const message)

{

int value = 0;

printf("%s", message);

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

if (result != 1)

{

errno = EIO;

perror("Error :");

abort();

}

return value;

}

size\_t GetArraySize()

{

int size = GetInt("Enter size of array: ");

if (size <= 0)

{

errno = ERANGE;

perror("Error :");

abort();

}

return (size\_t)size;

}

int\* GetArray( const size\_t size)

{

int\* a = malloc(size \* sizeof(int));

if (NULL == a)

{

errno = ENOMEM;

perror("Error :");

abort();

}

return a;

}

void FillArray(int\* a, const size\_t size)

{

puts("Enter array elements: ");

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

{

a[i] = GetInt("Input element = ");

}

}

void FillArrayRand(int\* a, const size\_t size)

{

puts("Enter array elements: ");

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

{

a[i] = rand() % 201 - 100; // диапазон случайных чисел от -100 до 100

}

}

void PrintArray(int\* a, const size\_t size)

{

puts("Array elements: ");

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

{

printf("Element %zu = %d\n", i, a[i]);

}

}

int\* getCpyArray(const size\_t size, const int\* const a)

{

int\* resultArray = GetArray(size);

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

{

resultArray[i] = a[i];

}

return resultArray;

}

void OpposiveValue(const size\_t size, int\*a )

{

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

{

a[i] = -1 \* a[i];

}

}

void Multiplicity(const size\_t size, int\* a)

{

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

{

if (a[i] % 3 == 0)

{

printf\_s("Element\_Multiplicity %d\n", i);

}

}

}

int SumValue (const size\_t size, int\* a)

{

int x = GetInt("Enter array elements:");

printf\_s("value %d\n", x);

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

{

if (a[i] + a[i + 1] == x) {

printf("The sum of the adjacent elements %d and %d is equal to %d\n", a[i], a[i + 1], x);

}

else {

printf("The sum of the adjacent elements % d and %d is not equal %d\n", a[i], a[i + 1], x);

}

}

}

void FreeArray(int\* array)

{

if (NULL != array)

{

free(array);

}

}

* 1. Решение тестового примера

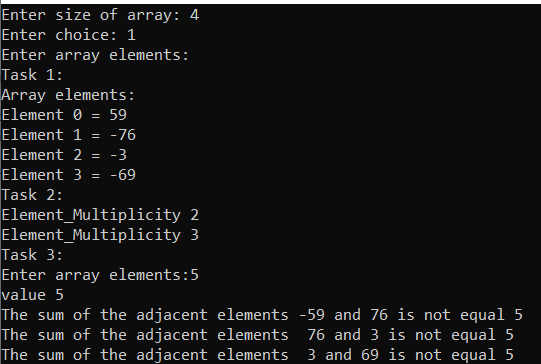


Рисунок 13 – Решение тестового примера

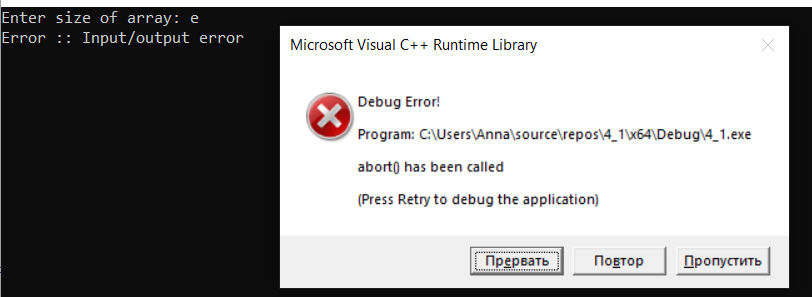


Рисунок 14 – Вывод программы, когда введена – буква

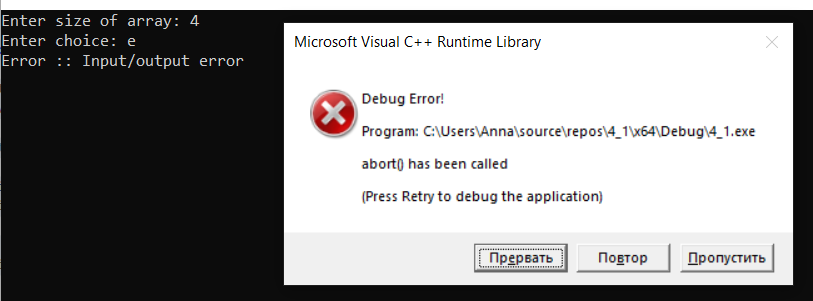


Рисунок 15 – Вывод программы, когда введена – буква

* 1. Зачёт задания в GitHub

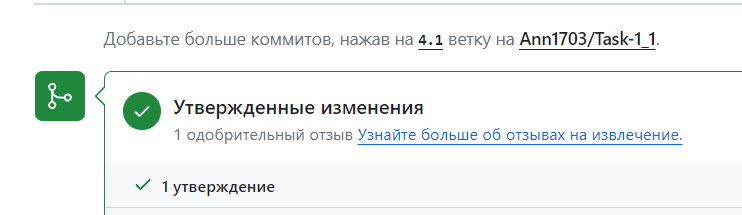


Рисунок 16 – Зачёт задания в GitHub

# Задание 4.2

* 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 2 – Формулировка задания 3.2

|  |  |  |
| --- | --- | --- |
| **Вар** | **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] |

* 1. Блок-схема алгоритма

Блок-схема алгоритма представлена на рисунках (Рисунки 17-28)

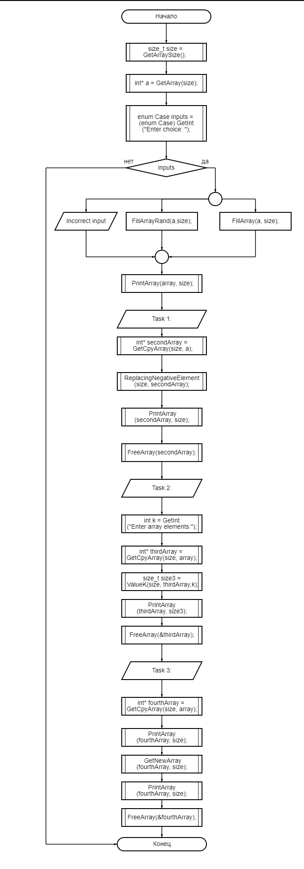


Рисунок 17 – Блок-схема функции main()

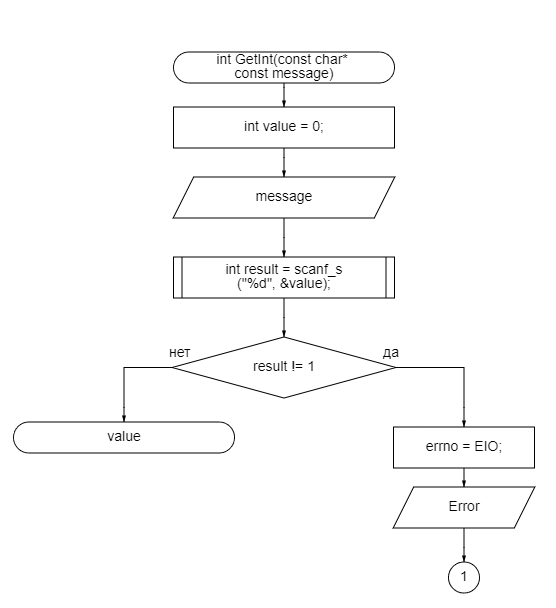


Рисунок 18 – Блок-схема функции GetInt ()

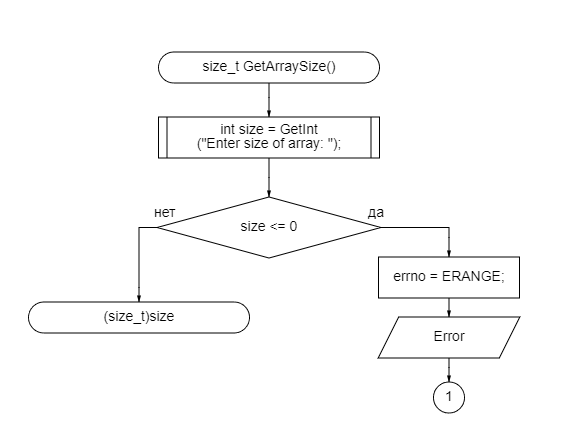


Рисунок 19 – Блок-схема к функции GetArraySize()

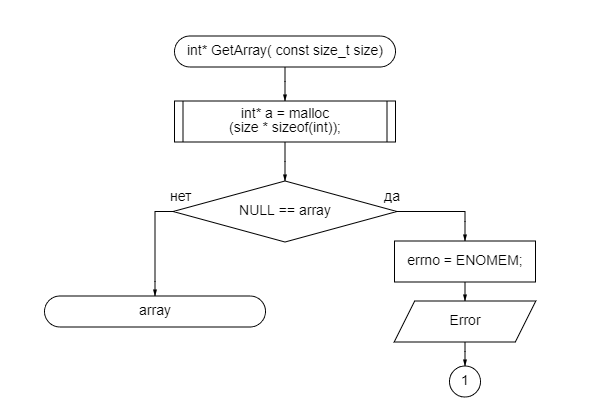


Рисунок 20 – Блок-схема функции GetArray ()

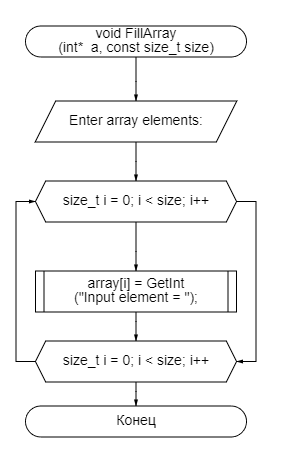


Рисунок *21* – Блок-схема функции FillArray ()

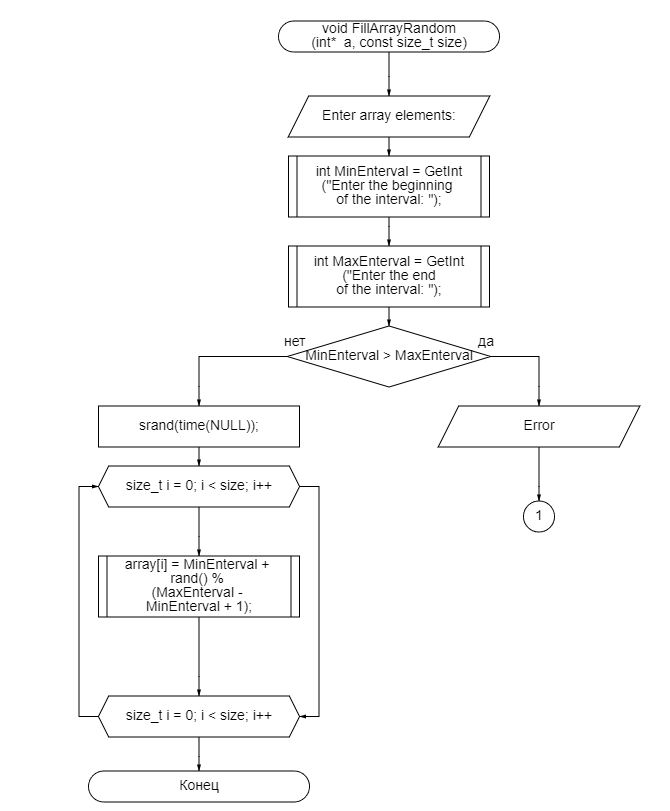


Рисунок 22– Блок-схема функции FillArrayRandom ()

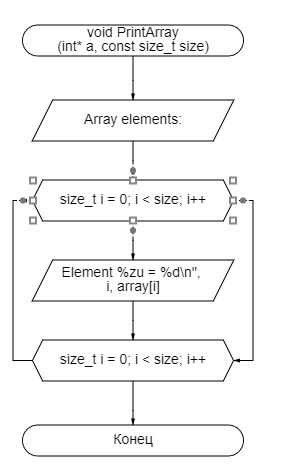


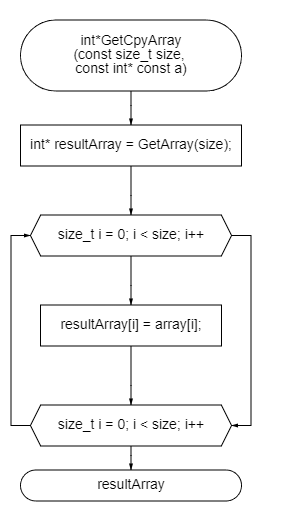
Рисунок 23– Блок-схема функции PrintArray ()

Рисунок 24– Блок-схема функции getCpyArray ()

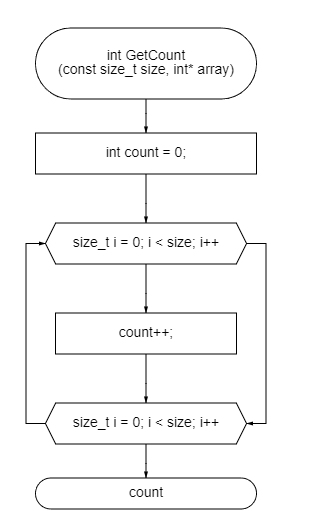


Рисунок 25– Блок-схема функции GetCount ()

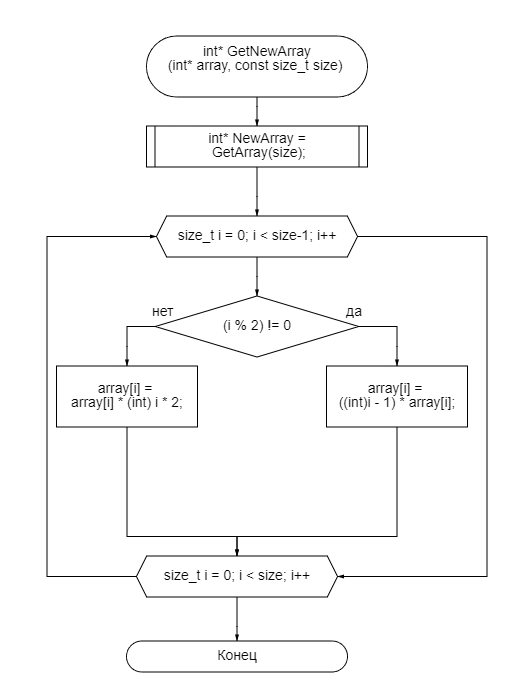


Рисунок 26– Блок-схема функции GetNewArray ()

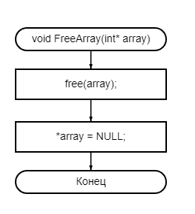


Рисунок 27– Блок-схема функции FreeArray ()

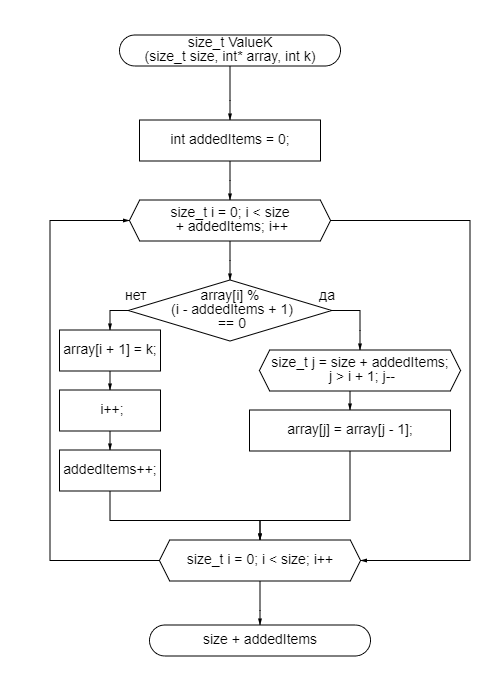


Рисунок 28– Блок-схема функции ValueK ()

Код задания 4.2

#include <stdio.h>

#include <errno.h>

#include <stdlib.h>

/\*\*

\* @brief считывает и проверяет ввод переменной типа int

\* @return Возвращает считанное значение

\*/

int GetInt(const char\* const message);

/\*\*

\* @brief функция заменяет первый отрицательный элемент массива нулем.

\* @param size - размер массива

\* @param a - массив

\*/

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

/\*\*

\* @brief фунция подставляет число К после всех элементов, кратных своему номеру

\* @param size - размер массива

\* @param a - массив

\*/

size\_t ValueK(size\_t size, int\* array, int\* count);

/\*\*

\* @brief подсичтывает количество элементов в массиве

\* @param count - переменная в которую записывает сколько элементов в массиве

\* @return count - возвращает количество элементов в массиве

\*/

int GetCount(const size\_t size, int\* array);

/\*\*

\* @brief функция печатает массив

\* @param size - размер массива

\* @param a - массив

\*/

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

/\*\*

\* @brief функция для создания массива

\* @param а - название массива

\* @param size - размер массива

\* @return возвращает массив

\*/

int\* GetArray(const size\_t size);

/\*\*

\* @brief функция для 3 пукнта задания

\* @param size - размер массива

\* @param a - массив

\* @param d - новый массив

\* @return возвращает новый массив

\*/

int\* GetNewArray(int\* array, const size\_t size);

/\*\*

\* @brief функция для заполнения массива в ручную

\* @param size - размер массива

\* @param a - массив

\*/

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

/\*\*

\* @brief функция для заполнения массива в рандомными числами

\* @param size - размер массива

\* @param a - массив

\* @param MinEnterval - минимальное значение интервала для генерации чисел

\* @param MaxEnterval - максимальное значение интервала для генерации чисел

\*/

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

/\*\*

\* @brief функция очищает массив

\* @param a - массив

\*/

void FreeArray(int\* array);

/\*\*

\* @brief функция копирует массив

\* @param resultArray - копия массива

\* @param size - размер массива

\* @param a - массив

\* @return resultArray - возвращает копию массива

\*/

int\* GetCpyArray(const size\_t size, const int\* const array);

/\*\*

\* @brief функция для определения размера массива

\* @return возвращает размер массива

\*/

size\_t GetArraySize();

/\*\*

\* @brief Выбор заполнения массива.

\*/

enum Manual

{

random = 1,

enter = 2,

};

int main()

{

size\_t size = GetArraySize();

int\* array = GetArray(size);

enum Manual inputs = (enum Manual)GetInt("Enter choice: ");

switch (inputs)

{

case random:

FillArrayRandom(array, size);

break;

case enter:

FillArray(array, size);

break;

default:

puts("Incorrect input.\n");

return 1;

}

PrintArray(array, size);

puts("Task 1:");

int\* secondArray = GetCpyArray(size, array);

ReplacingNegativeElement(size, secondArray);

PrintArray(secondArray, size);

FreeArray(&secondArray);

puts("Task 2:");

int k = GetInt("Enter array elements:");

int\* thirdArray = GetCpyArray(size, array);

size\_t size3 = ValueK(size, thirdArray,k);

PrintArray(thirdArray, size3);

FreeArray(&thirdArray);

puts("\nTask 3:");

int\* fourthArray = GetCpyArray(size, array);

PrintArray(fourthArray, size);

GetNewArray(fourthArray, size);

PrintArray(fourthArray, size);

FreeArray(&fourthArray);

return 0;

}

int GetInt(const char\* const message)

{

int value = 0;

printf("%s", message);

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

if (result != 1)

{

errno = EIO;

perror("Error :");

abort();

}

return value;

}

size\_t GetArraySize()

{

int size = GetInt("Enter size of array: ");

if (size <= 0)

{

errno = ERANGE;

perror("Error :");

abort();

}

return (size\_t)size;

}

int\* GetArray(const size\_t size)

{

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

if (NULL == array)

{

errno = ENOMEM;

perror("Error :");

abort();

}

return array;

}

void FillArray(int\* array, const size\_t size)

{

puts("Enter array elements: ");

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

{

array[i] = GetInt("Input element = ");

}

}

void FillArrayRandom(int\* array, const size\_t size)

{

puts("Enter array elements: ");

int MinEnterval = GetInt("Enter the beginning of the interval: ");

int MaxEnterval = GetInt("Enter the end of the interval : ");

if (MinEnterval > MaxEnterval)

{

perror("Error: ");

abort();

}

srand(time(NULL));

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

{

array[i] = MinEnterval + rand() % (MaxEnterval - MinEnterval + 1);

}

}

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

{

puts("Array elements: ");

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

{

printf("Element %zu = %d\n", i, array[i]);

}

}

int\* GetCpyArray(const size\_t size, const int\* const array)

{

int\* resultArray = GetArray(size);

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

{

resultArray[i] = array[i];

}

return resultArray;

}

void ReplacingNegativeElement(const size\_t size, int\* array)

{

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

if (array[i] < 0)

{

array[i] = 0;

break;

}

}

}

int GetCount(const size\_t size, int\* array)

{

int count = 0;

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

{

count++;

}

return count;

}

size\_t ValueK(size\_t size, int\* array, int k)

{

int addedItems = 0;

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

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

for (size\_t j = size + addedItems; j > i + 1; j--) {

array[j] = array[j - 1];

}

array[i + 1] = k;

i++;

addedItems++;

}

}

return size + addedItems;

}

int\* GetNewArray(int\* array, const size\_t size)

{

int\* NewArray = GetArray(size);

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

{

if ((i % 2) != 0)

{

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

}

else

{

array[i] = array[i] \* (int) i \* 2;

}

}

return array;

}

void FreeArray(int\*\* array)

{

free(\*array);

\*array = NULL;

}

* 1. Решение тестового примера

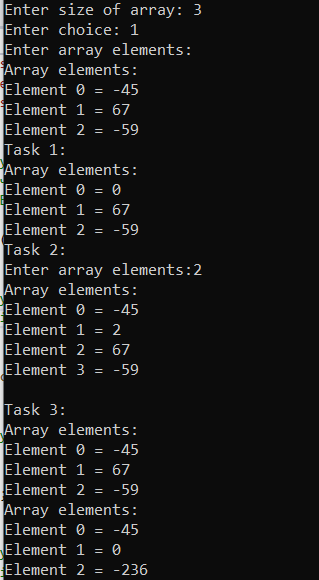


Рисунок 29 – Решение тестового примера

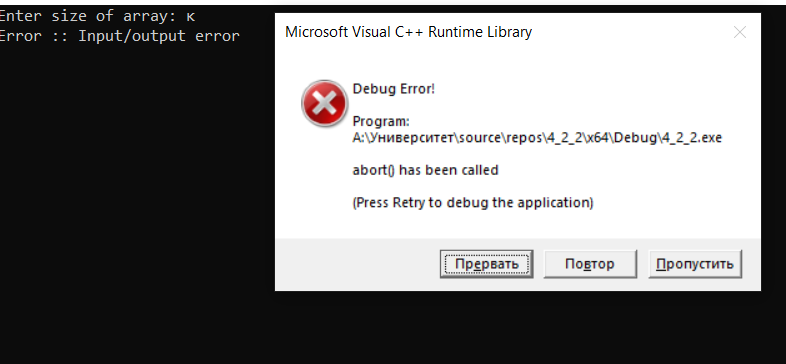


Рисунок 30 – Вывод программы, когда введена буква

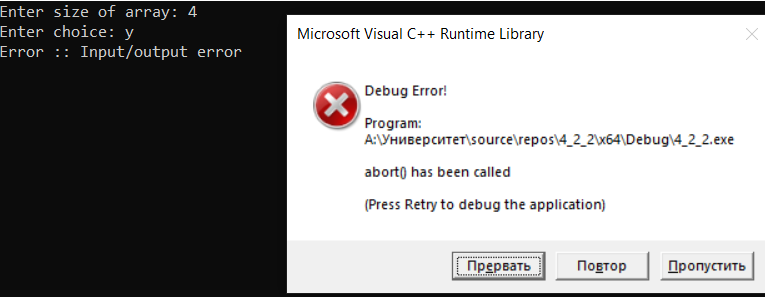


Рисунок 31 – Вывод программы, когда введена буква

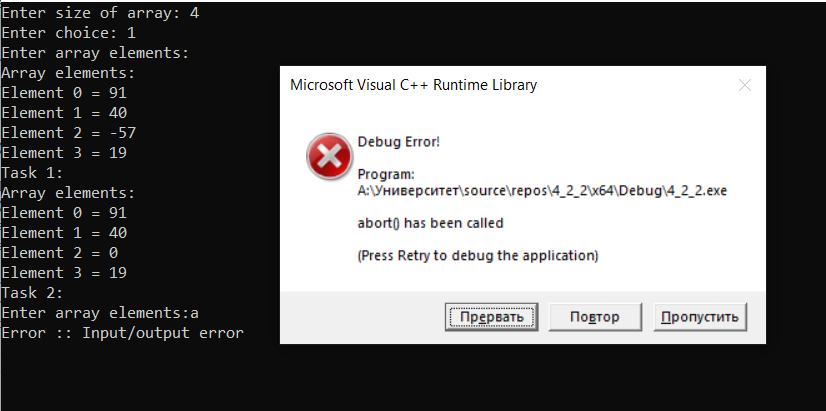


Рисунок 32 – Вывод программы, когда введена буква

* 1. Зачёт задания в GitHub

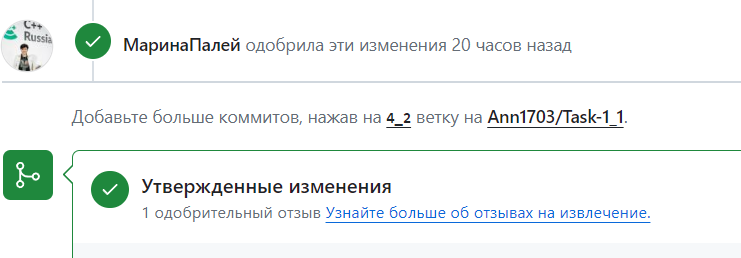


Рисунок 33 – Зачёт задания в GitHub

# Задание 4.3

* 1. Формулировка задания

Создать многомерный массив nˣm 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.

Таблица 3 – Формулировка задания 3.3

|  |  |
| --- | --- |
| Вар | Задачи |
| 5 | 1. Заменить максимальный элемент каждой строки на противоположный по знаку. 2. Вставить после всех столбцов, содержащих максимальный элемент столбец из нулей. |

* 1. Блок-схема алгоритма

Блок-схема алгоритма представлена на рисунках (Рисунки 34-42)

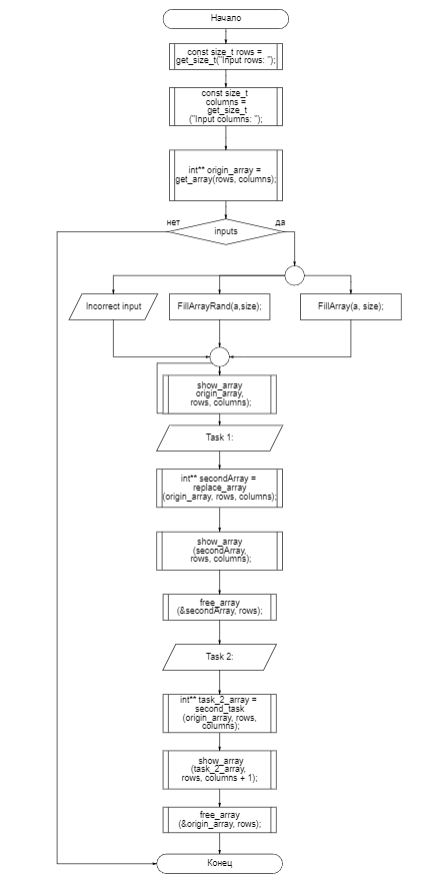


Рисунок 34 – Блок-схема функции main()

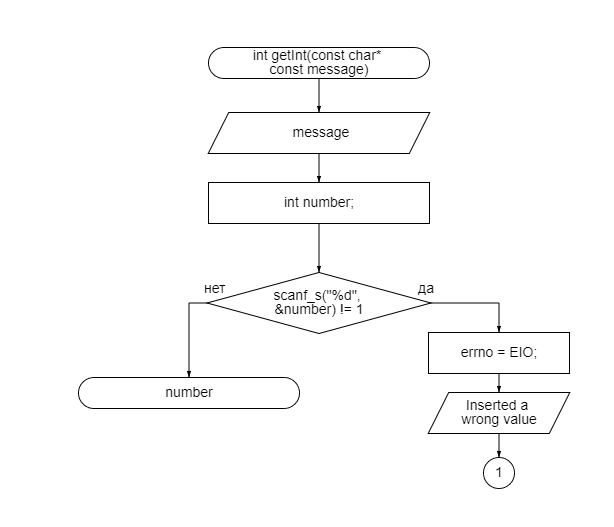


Рисунок 35 – Блок-схема функции getInt()

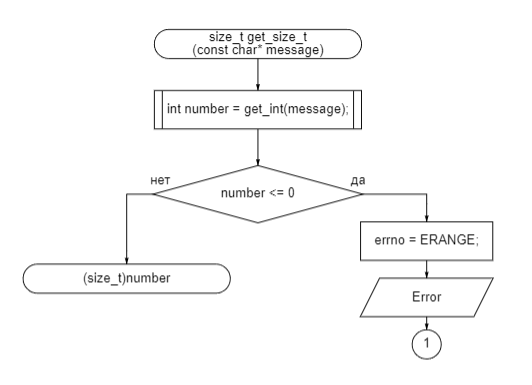


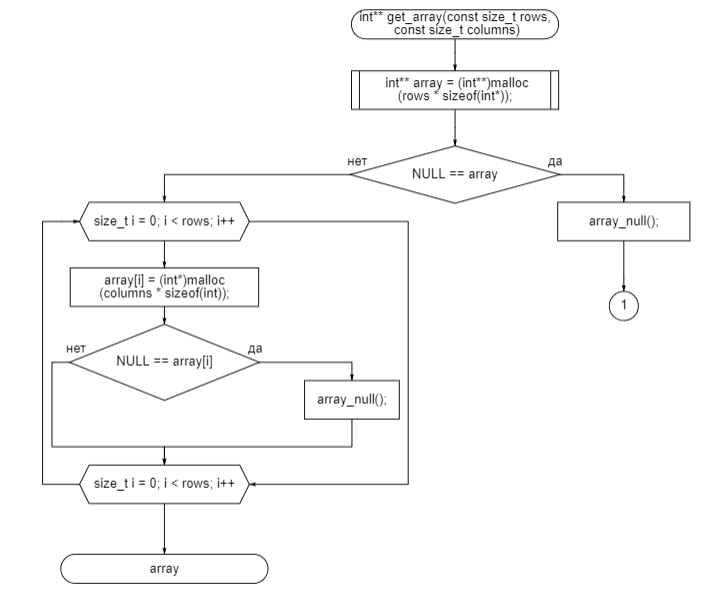
Рисунок 36 – Блок-схема функции get\_size\_t () 

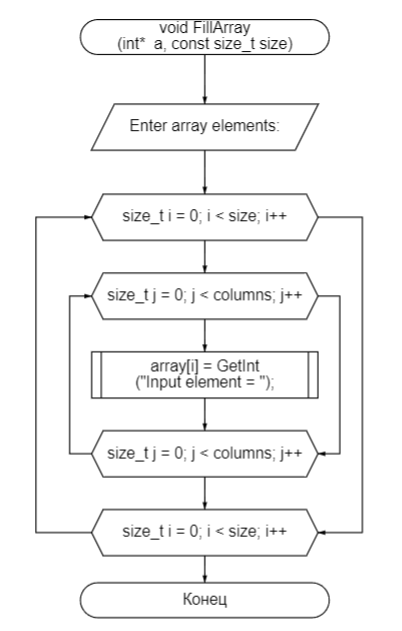
Рисунок 37 – Блок-схема функции get\_array ()

Рисунок 38 – Блок-схема функции FillArray ()

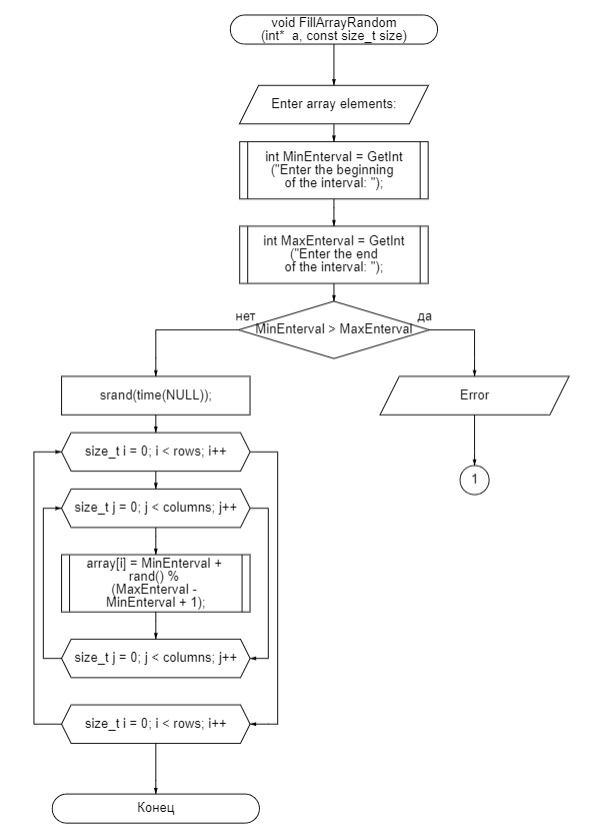


Рисунок 39– Блок-схема функции FillArrayRandom ()

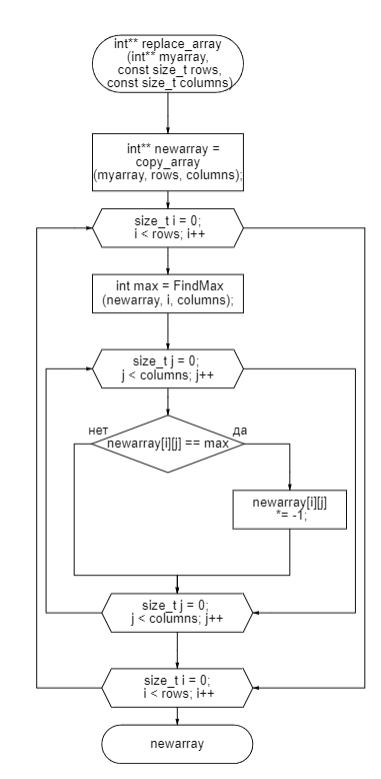


Рисунок 40 – Блок-схема функции replace\_array ()

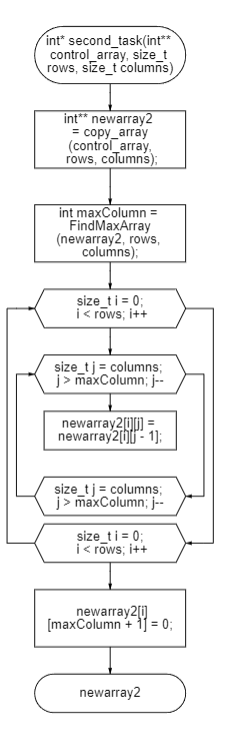


Рисунок 41 – Блок-схема функции second\_task ()

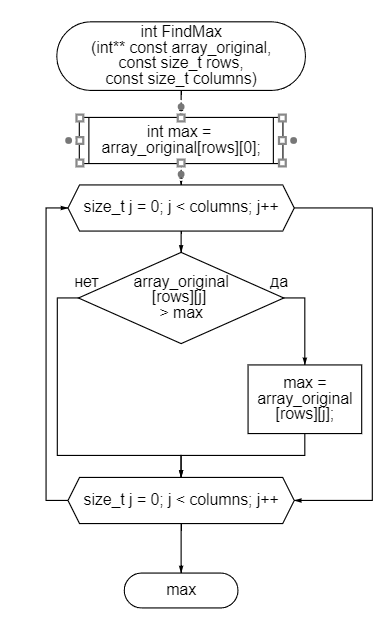


Рисунок 42 – Блок-схема функции FindMax ()

* 1. Код задания 4.3

#include <stdio.h>

#include <stdlib.h>

#include <errno.h>

/\*\*

\* @brief считывает и проверяет ввод переменной типа int

\* @return Возвращает считанное значение

\*/

int get\_int(const char\* message);

/\*\*

\* @brief функция для определения размера массива

\* @return возвращает размер массива

\*/

size\_t get\_size\_t(const char\* message);

/\*\*

\* @brief функция для создания массива

\* @param size - размер массива

\* @param array - массив

\* @param rows - строка массива

\* @param columns - столбец массива

\*/

int\*\* get\_array(const size\_t rows, const size\_t columns);

/\*\*

\* @brief Функция для обработки ошибки выделения памяти

\*/

void array\_null();

/\*\*

\* @brief Функция для заполнения массива в ручную

\* @param array - массив

\* @param rows - строка массива

\* @param columns - столбец массива

\*/

void fill\_array(int\*\* const array, const size\_t rows, const size\_t columns);

/\*\*

\* @brief функция для заполнения массива в рандомными числами

\* @param array - массив

\* @param rows - строка массива

\* @param columns - столбец массива

\* @param MinEnterval - минимальное значение интервала для генерации чисел

\* @param MaxEnterval - максимальное значение интервала для генерации чисел

\*/

void fill\_array\_random(int\*\* const array, const size\_t rows, const size\_t columns);

/\*\*

\* @brief функция печатает массив

\* @param size - размер массива

\* @param array - массив

\* @param rows - строка массива

\* @param columns - столбец массива

\*/

void show\_array(int\*\* const array, const size\_t rows, const size\_t columns);

/\*\*

\* @brief функция вставляет после всех столбцов, содержащих максимальный элемент столбец из нулей

\* @param rows - строка массива

\* @param columns - столбец массива

\* @param maxColumn - максимальное значение матрицы

\*/

int\* second\_task(int\*\* control\_array, size\_t rows, size\_t cols);

/\*\*

\* @brief функция копирует массив

\* @param array\_copy - копия массива

\* @param rows - строка массива

\* @param columns - столбец массива

\* @param array\_original - массив

\*/

int\*\* copy\_array(int\*\* const array\_original, const size\_t rows, const size\_t columns);

/\*\*

\* @brief функция заменяет максимальный элемент каждой строки на противоположный по знаку

\* @param rows - строка массива

\* @param columns - столбец массива

\* @param array\_original - массив

\*/

int\*\* replace\_array(int\*\* myarray, const size\_t rows, const size\_t columns);

/\*\*

\* @brief функция ищет максимальный элемент в каждой строке

\* @param rows - строка массива

\* @param columns - столбец массива

\* @param array\_original - массив

\*/

int FindMax(int\*\* const array\_original, const size\_t rows, const size\_t columns);

/\*\*

\* @brief функция ищет максимальный элемент в каждой строке

\* @param rows - строка массива

\* @param columns - столбец массива

\* @param array\_original - массив

\*/

int\* FindMaxArray(int\*\* const array\_original, const size\_t rows, const size\_t columns);

/\*\*

\* @brief Функция для освобождения памяти, выделенной под массив

\* @param rows - строка массива

\* @param array - массив

\*/

void free\_array(int\*\* array, size\_t rows);

/\*\*

\* @brief Выбор заполнения массива.

\*/

enum Manual

{

random = 1,

enter = 2,

};

int main(void)

{

const size\_t rows = get\_size\_t("Input rows: ");

const size\_t columns = get\_size\_t("Input columns: ");

int\*\* origin\_array = get\_array(rows, columns);

enum Manual inputs = (enum Manual)get\_int("Enter choice: ");

switch (inputs)

{

case random:

fill\_array\_random(origin\_array, rows, columns);

break;

case enter:

fill\_array(origin\_array, rows, columns);

break;

default:

puts("Incorrect input.\n");

return 1;

break;

}

show\_array(origin\_array, rows, columns);

puts("Task1\n");

int\*\* secondArray = replace\_array(origin\_array, rows, columns);

show\_array(secondArray, rows, columns);

free\_array(&secondArray, rows);

puts("Task2");

int\*\* task\_2\_array = second\_task(origin\_array, rows, columns);

show\_array(task\_2\_array, rows, columns + 1);

free\_array(&origin\_array, rows);

return 0;

}

int get\_int(const char\* message)

{

puts(message);

int number;

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

{

errno = EIO;

perror("Inserted a wrong value\n");

abort();

}

return number;

}

size\_t get\_size\_t(const char\* message)

{

int number = get\_int(message);

if (number <= 0)

{

errno = ERANGE;

perror("Error: \n");

abort();

}

return (size\_t)number;

}

void array\_null()

{

errno = ENOMEM;

perror("Error: ");

abort();

}

int\*\* get\_array(const size\_t rows, const size\_t columns)

{

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

if (NULL == array)

{

array\_null();

}

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

{

array[i] = (int\*)malloc(columns \* sizeof(int));

if (NULL == array[i])

{

array\_null();

}

}

return array;

}

void fill\_array(int\*\* const array, const size\_t rows, const size\_t columns)

{

puts("Insert array elements:\n");

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

{

for (size\_t j = 0; j < columns; j++)

{

array[i][j] = get\_int("Get element");

}

}

}

void fill\_array\_random(int\*\* const array, const size\_t rows, const size\_t columns)

{

puts("Insert array elements:\n");

int MinEnterval = get\_int("Enter the beginning of the interval: ");

int MaxEnterval = get\_int("Enter the end of the interval : ");

if (MinEnterval > MaxEnterval)

{

perror("Error: ");

abort();

}

srand(time(NULL));

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

{

for (size\_t j = 0; j < columns; j++)

{

array[i][j] = MinEnterval + rand() % (MaxEnterval - MinEnterval + 1);

}

}

}

void show\_array(int\*\* const array, const size\_t rows, const size\_t columns)

{

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

{

for (size\_t j = 0; j < columns; j++)

{

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

}

printf("\n");

}

puts("\n");

}

void free\_array(int\*\*\* array, size\_t rows)

{

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

{

free(\*\*array);

\*\*array = NULL;

}

free(\*array);

\*array = NULL;

}

int\*\* copy\_array(int\*\* const array\_original, const size\_t rows, const size\_t columns)

{

int\*\* array\_copy = get\_array(rows, columns);

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

{

for (size\_t j = 0; j < columns; j++)

{

array\_copy[i][j] = array\_original[i][j];

}

}

return array\_copy;

}

int\*\* replace\_array(int\*\* myarray, const size\_t rows, const size\_t columns)

{

int\*\* newarray = copy\_array(myarray, rows, columns);

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

{

int max = FindMax(newarray, i, columns);

for (size\_t j = 0; j < columns; j++)

{

if (newarray[i][j] == max)

{

newarray[i][j] \*= -1;

}

}

}

return newarray;

}

int\* second\_task(int\*\* control\_array, size\_t rows, size\_t columns)

{

int\*\* newarray2 = copy\_array(control\_array, rows, columns);

int maxColumn = FindMaxArray(newarray2, rows, columns);

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

for (size\_t j = columns; j > maxColumn; j--)

{

newarray2[i][j] = newarray2[i][j - 1];

}

newarray2[i][maxColumn + 1] = 0;

}

return newarray2;

}

int FindMax(int\*\* const array\_original, const size\_t rows, const size\_t columns)

{

int max = array\_original[rows][0];

for (size\_t j = 1; j < columns; j++)

{

if (array\_original[rows][j] > max)

{

max = array\_original[rows][j];

}

}

return max;

}

int\* FindMaxArray(int\*\* array\_original, const size\_t rows, const size\_t columns)

{

int maxElement = array\_original[0][0];

int maxColumn = 0;

for (int j = 0; j < columns; j++)

{

for (int i = 0; i < rows; i++)

{

if (array\_original[i][j] > maxElement)

{

maxElement = array\_original[i][j];

maxColumn = j;

}

}

}

return maxColumn;

}

* 1. Решение тестового примера

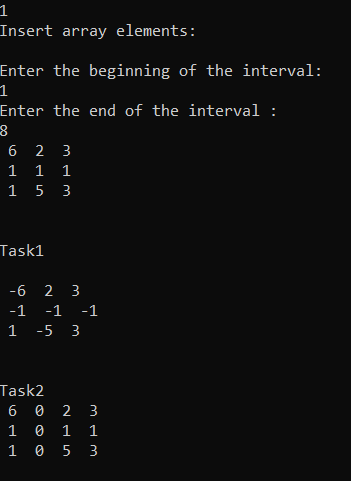


Рисунок 43 – Решение тестового примера

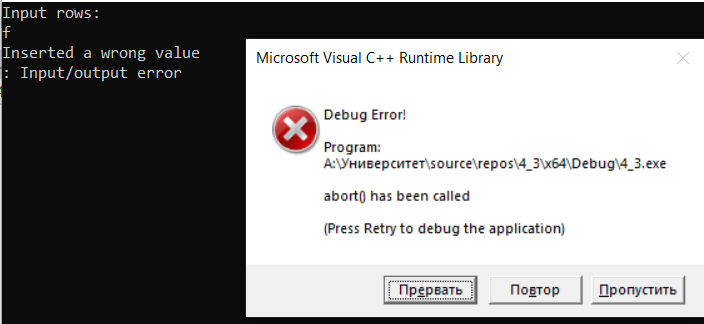


Рисунок 44 – Вывод программы, когда введена буква

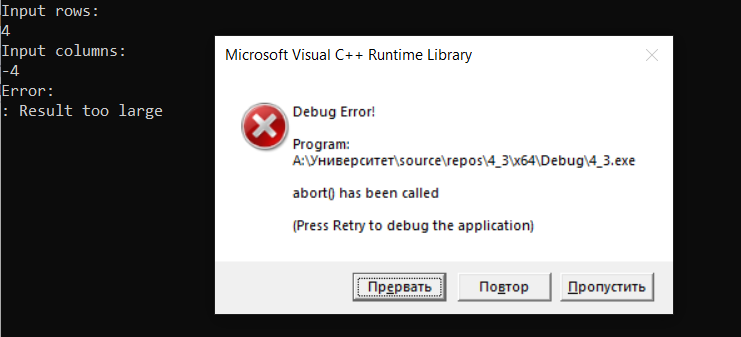


Рисунок 45 – Вывод программы, когда введено отрицательное значение

* 1. Зачёт задания в GitHub

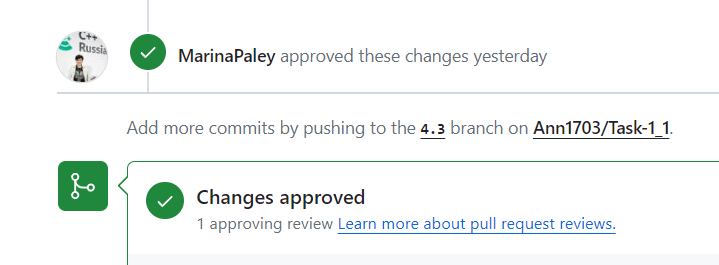


Рисунок 46 – Зачёт задания в GitHub