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 11

Выполнил: ст. гр. ТКИ-142

Луканкин Вячеслав Сергеевич

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

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

Москва 2023

Оглавление

**Задание 4.11**

1.1 ФОРМУЛИРОВКА ЗАДАНИЯ 1

1.2 БЛОК-СХЕМА АЛГОРИТМА2-10

1.3 ТЕКСТ ПРОГРАММЫ НА ЯЗЫКЕ C11-17

1.4 РЕЗУЛЬТАТ ВЫПОЛНЕНИЯ ПРОГРАММЫ18

1.5 ВЫПОЛНЕНИЕ ТЕСТОВЫХ ПРИМЕРОВ19-21

1.6 ОТМЕТКА О ВЫПОЛНЕНИИ ЗАДАНИЯ В ВЕБ-ХОСТИНГЕ СИСТЕМЫ КОНТРОЛЯ ВЕРСИЙ22

**Задание 4.2** 23

1.1 ФОРМУЛИРОВКА ЗАДАНИЯ 23

1.2 БЛОК-СХЕМА АЛГОРИТМА24-39

1.3 ТЕКСТ ПРОГРАММЫ НА ЯЗЫКЕ C40-52

1.4 РЕЗУЛЬТАТ ВЫПОЛНЕНИЯ ПРОГРАММЫ53

1.5 ВЫПОЛНЕНИЕ ТЕСТОВЫХ ПРИМЕРОВ54

1.6 ОТМЕТКА О ВЫПОЛНЕНИИ ЗАДАНИЯ В ВЕБ-ХОСТИНГЕ СИСТЕМЫ КОНТРОЛЯ ВЕРСИЙ55

**Задание 4.3 56**

1.1 ФОРМУЛИРОВКА ЗАДАНИЯ 56

1.2 БЛОК-СХЕМА АЛГОРИТМА57-66

1.3 ТЕКСТ ПРОГРАММЫ НА ЯЗЫКЕ C67-80

1.4 РЕЗУЛЬТАТ ВЫПОЛНЕНИЯ ПРОГРАММЫ81

1.5 ВЫПОЛНЕНИЕ ТЕСТОВЫХ ПРИМЕРОВ82

1.6 ОТМЕТКА О ВЫПОЛНЕНИИ ЗАДАНИЯ В ВЕБ-ХОСТИНГЕ СИСТЕМЫ КОНТРОЛЯ ВЕРСИЙ83

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), указанные в таблице (Таблица 1). Составить блок-схему.

Таблица  – Исходные данные

|  |  |  |
| --- | --- | --- |
| Вариант | Задачи | Интервал |
| 11 | 1. Найти сумму четных элементов, значения которых больше заданного числа. 2. Умножить все четные положительные элементы на последний элемент массива. 3. Найти номер первой пары соседних элементов с разными знаками. | [-100;200] |

* 1. Блок-схема алгоритма

Блок-схема основного алгоритма представлена ниже (Рисунок 1, Рисунок 2, Рисунок 3, Рисунок 4, Рисунок 5). Блок-схемы вспомогательных функций представлены ниже (Рисунок 6, Рисунок 7, Рисунок 8, Рисунок 9, Рисунок 10, Рисунок 11, Рисунок 12, Рисунок 13, Рисунок 14, Рисунок 15, Рисунок 16, Рисунок 17)

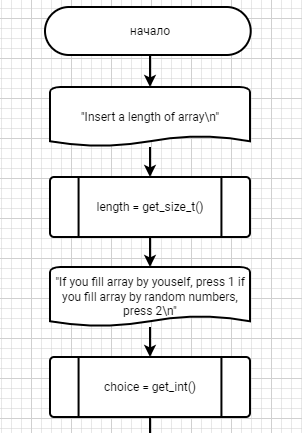


Рисунок  ­ Блок-схема основного алгоритма часть 1

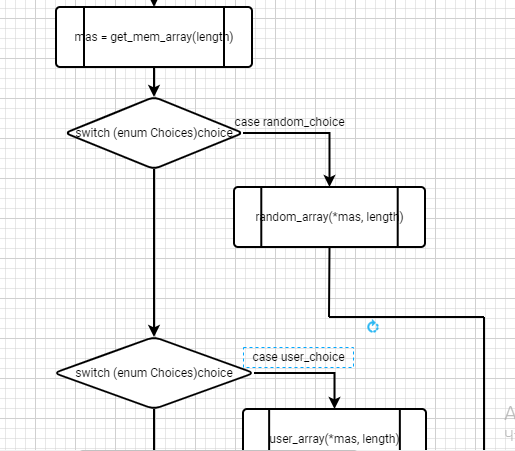


Рисунок  – Блок-схема основного алгоритма часть 2

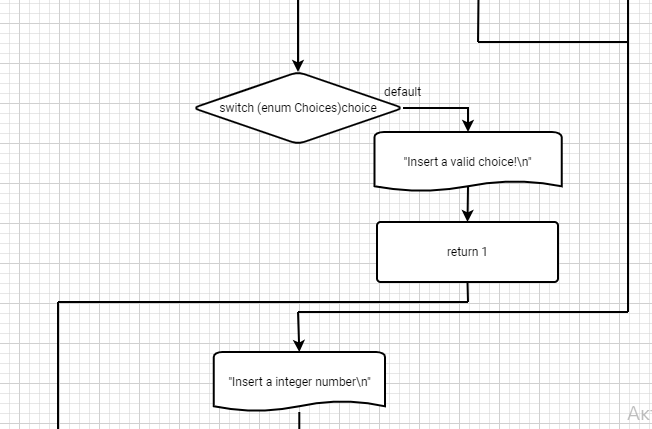


Рисунок  – Блок-схема основного алгоритма часть 3

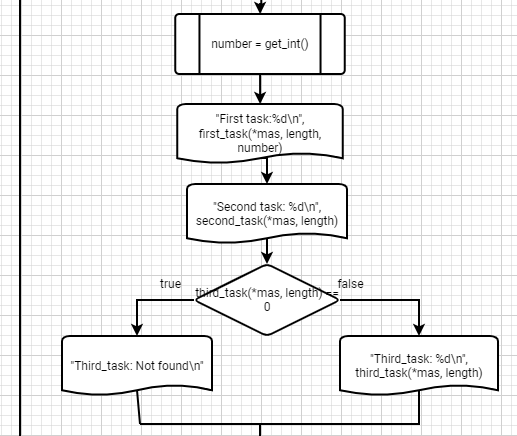


Рисунок  – Блок-схема основного алгоритма часть 4

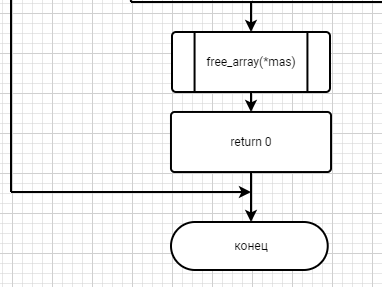


Рисунок  – Блок-схема основного алгоритма часть 5

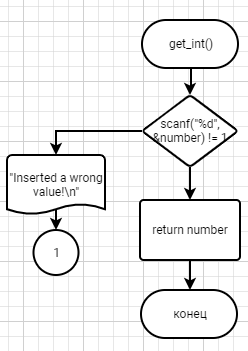


Рисунок  – Блок-схема функции get\_int()

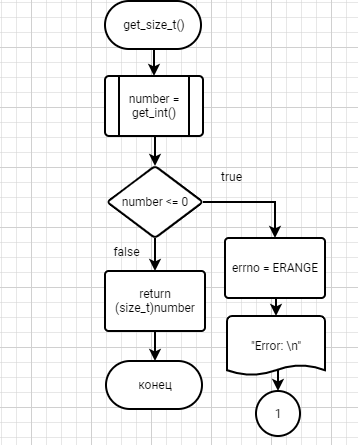


Рисунок  – Блок-схема функции get\_size\_t()

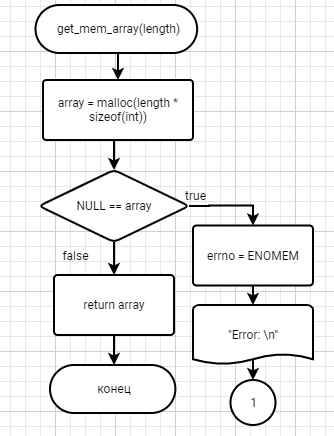


Рисунок 8 – Блок-схема функции get\_mem\_array(length)

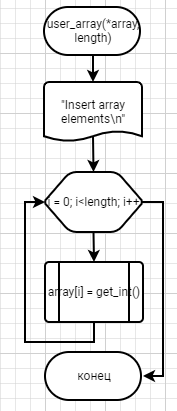


Рисунок 9 – Блок-схема функции user\_array(\*array, length)

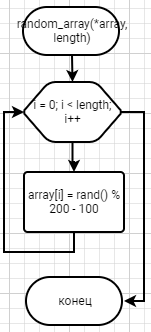


Рисунок 10 – Блок-схема функции random\_array(\*array, length)

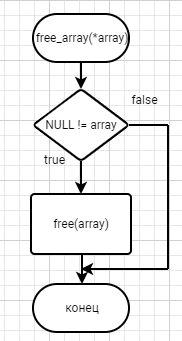


Рисунок  – Блок-схема функции free\_array(\*array)

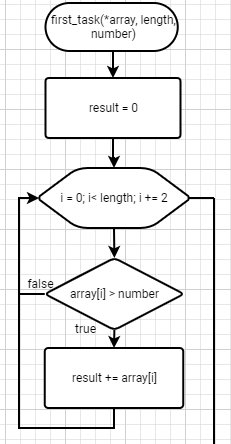


Рисунок 12 – Блок-схема функции first\_task(\*array, length, number) часть 1

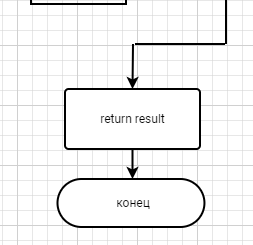


Рисунок 13 – Блок-схема функции first\_task(\*array, length, number) часть 2

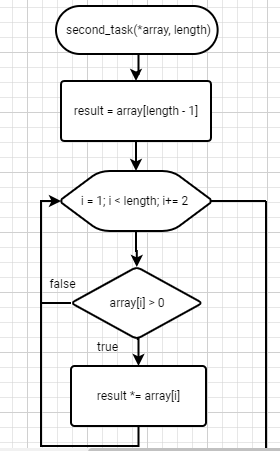


Рисунок 14 – Блок-схема функции second\_task(\*array, length) часть 1

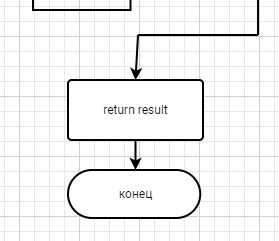


Рисунок 15 – Блок-схема функции second\_task(\*array, length) часть 2

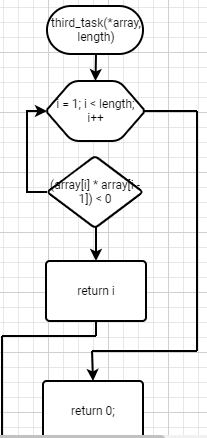


Рисунок 16 – Блок-схема функции third\_task(\*array, length) часть 1

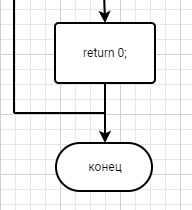


Рисунок 17 – Блок-схема функции third\_task(\*array, length) часть 2

* 1. Текст программы на языке C

#include <stdio.h>

#include <stdlib.h>

#include <malloc.h>

#include <errno.h>

/\*\*

\* @brief Функция присваивает целочисленное значение переменной

\* @return целочисленная цифра

\*/

int get\_int();

/\*\*

\* @brief Функция присваивает переменной целочисленное значение и проверяет его на положителность

\* @return number положительное число

\*/

size\_t get\_size\_t();

/\*\*

\* @brief структура хранит константы, указывающие выбор пользователя заполнить массив

\* @brief user\_choice хранит значение, вызывающее ввод массива вручную

\* @brief random\_choice хранит значение, вызывающее ввод массива случайными числами

\*/

enum Choices

{

user\_choice = 1,

random\_choice = 2,

};

/\*\*

\* @brief Функция выделяет память под массив

\* @param length длина массива

\* @return array указатель на пустой массив

\*/

int\* get\_mem\_array(const int length);

/\*\*

\* @brief Функция заполняет массив, элементы вводит пользователь

\* @param array указатель на заполняемый массив

\* @param length длина массива

\*/

void user\_array(int\* const array, const size\_t length);

/\*\*

\* @brief Функция заполняет массив рандомными числами в диапазоне [-100:200]

\* @param array указатель на заполняемый массив

\* @param length длина массива

\*/

void random\_array(int\* const array, const size\_t length);

/\*\*

\* @brief Находит сумму четных элементов, значения которых больше заданного числа.

\* @param array указатель на массив

\* @param length размерность массива

\* @param number Заданное число

\* @return result результат

\*/

int first\_task(int\* const array, size\_t length, const int number);

/\*\*

\* @brief Умножает все четные положительные элементы на последний элемент массива.

\* @param array указатель на массив

\* @param length размерность массива

\* @return result результат

\*/

int second\_task(int\* const array, size\_t length);

/\*\*

\* @brief Находит номер первой пары соседних элементов с разными знаками.

\* @param array указатель на массив

\* @param length размерность массива

\* @return result результат

\*/

int third\_task(int\* const array, const int length);

/\*\*

\* @brief Функция, освобождающая массив

\* @param array указатель на массив

\*/

void free\_array(int\* array);

/\*\*

\* @brief Точка входа в программу

\* @return 0 Программа исправна

\* @return 1 Программа досрочно закрылась с ошибкой

\*/

int main()

{

puts("insert a length of array\n");

size\_t length = get\_size\_t();

puts("if you fill array by youself, press 1, if you fill array by random numbers, press 2\n");

int choice = get\_int();

int\* mas = get\_mem\_array(length);

switch ((enum Choices)choice)

{

case random\_choice:

random\_array(\*mas, length);

break;

case user\_choice:

user\_array(\*mas, length);

break;

default:

puts("Insert a valid choice!\n");

return 1;

}

puts("insert a integer number\n");

int number = get\_int();

printf("First task: %d\n", first\_task(\*mas, length, number));

printf("Second task: %d\n", second\_task(\*mas, length));

if (third\_task(\*mas, length) == 0)

{

puts("Third task: Not found\n");

}

else

{

printf("Third task: %d", third\_task(\*mas, length));

}

free\_array(\*mas);

return 0;

}

int get\_int()

{

int number;

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

{

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

abort();

}

return number;

}

size\_t get\_size\_t()

{

int number = get\_int();

if (number <= 0)

{

errno = ERANGE;

perror("Error: \n");

abort();

}

return (size\_t)number;

}

int\* get\_mem\_array(const int length)

{

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

if (NULL == array)

{

errno = ENOMEM;

perror("Error: ");

abort();

}

return array;

}

void user\_array(int\* const array, const size\_t length)

{

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

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

{

array[i] = get\_int();

}

}

void random\_array(int\* const array, const size\_t length)

{

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

{

array[i] = rand() % 200 - 100;

}

}

int first\_task(int\* const array, size\_t length, const int number)

{

int result = 0;

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

{

result += array[i];

}

return result;

}

int second\_task(int\* const array, size\_t length)

{

int result = array[length-1];

for (size\_t i = 1; i < length-1; i += 2)

{

if (array[i] > 0)

{

result \*= array[i];

}

}

return result;

}

int third\_task(int\* const array, const int length)

{

for (int i = 1; i < length; i++)

{

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

{

return i;

}

}

return 0;

}

void free\_array(int\* array)

{

if (NULL != array)

{

free(array);

}

}

* 1. Результаты выполнения программы

Результаты выполнения программы представлены ниже (Рисунок 18).

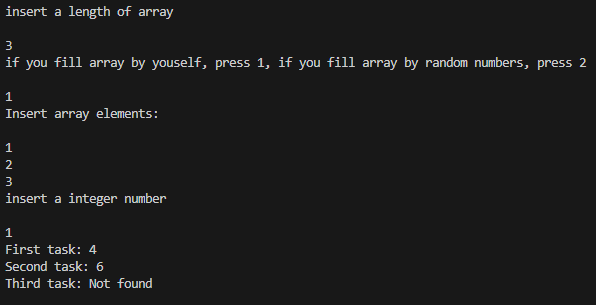


Рисунок  – Результат выполнения программы

* 1. Выполнение тестовых примеров

В программе MS Excel выполнены тестовые примеры. Результаты их выполнения представлены ниже (Рисунок 19, Рисунок 20, Рисунок 21, Рисунок 22, Рисунок 23, Рисунок 24).

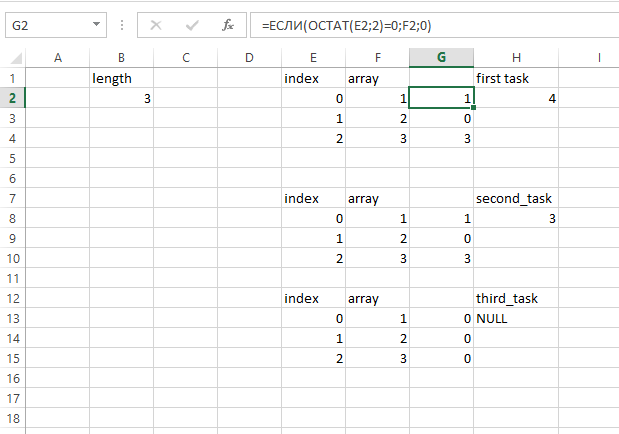


Рисунок  – Выбор подходящих элементов под функцию first\_task

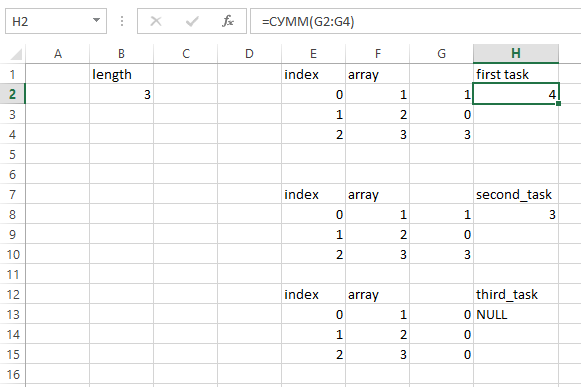


Рисунок  – Результат расчета функции first\_task

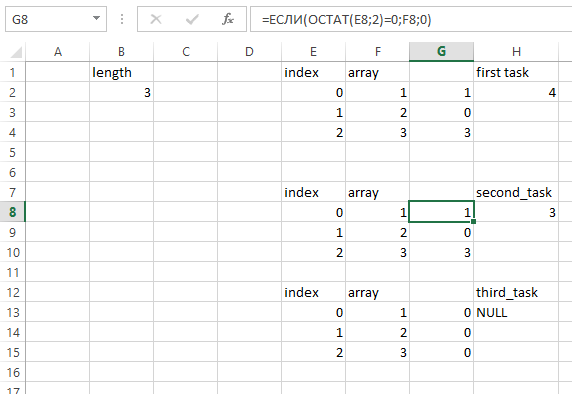


Рисунок  –Выбор подходящих элементов из массива функции second\_task

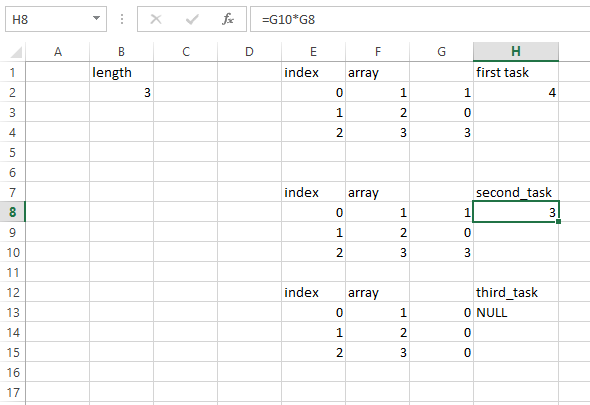


Рисунок  –Результат расчета функции second\_task

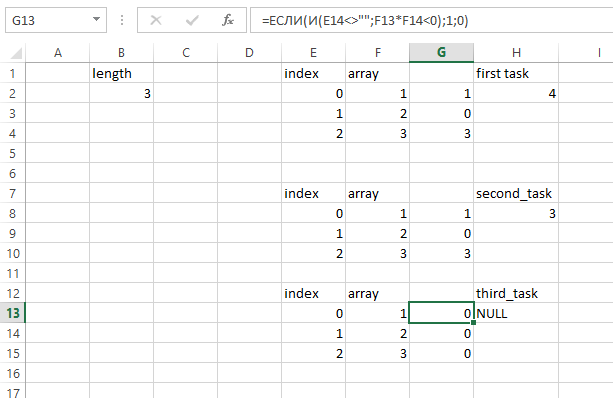


Рисунок  – Выбор подходящих элементов под функцию third\_task

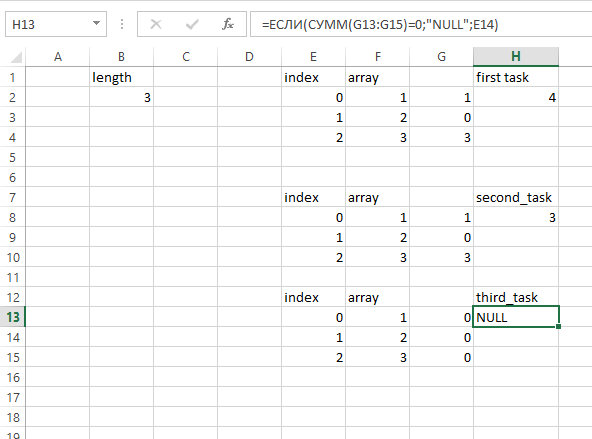
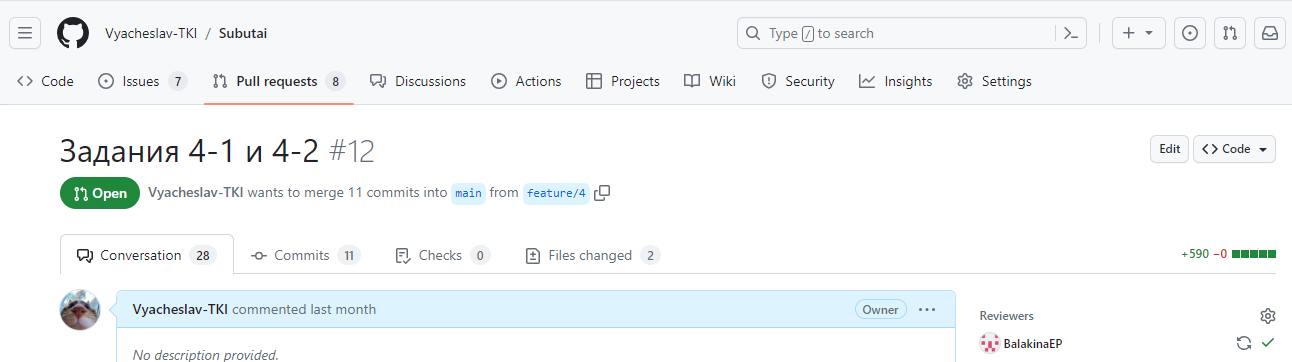


Рисунок  – Результат расчета функции third\_task

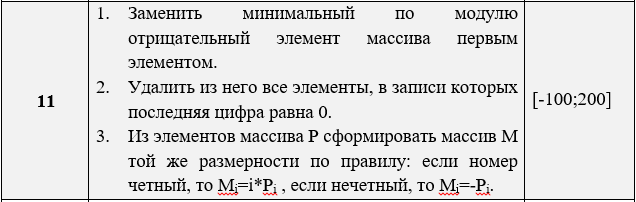
* 1. Отметка о выполнении задания в веб-хостинге системы контроля версий



1. 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).Данные записаны в таблице (Таблица 1). Вывести массив на экран. Составить блок-схему.

Таблица  – Исходные данные



2.2 Блок-схема алгоритма

Блок-схема основного алгоритма представлена ниже (Рисунок 25, Рисунок 26, Рисунок 27, Рисунок 28, Рисунок 29). Блок-схемы вспомогательных функций представлены ниже (Рисунок 30, Рисунок 31, 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 44, Рисунок 45, Рисунок 46, Рисунок 47, Рисунок 48, Рисунок 49, Рисунок 50, Рисунок 51, Рисунок 52).

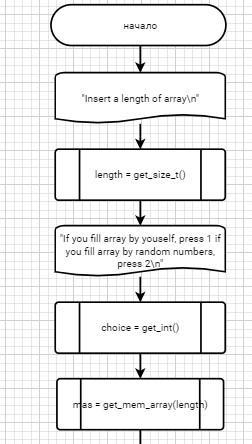


Рисунок  ­ Блок-схема основного алгоритма часть 1

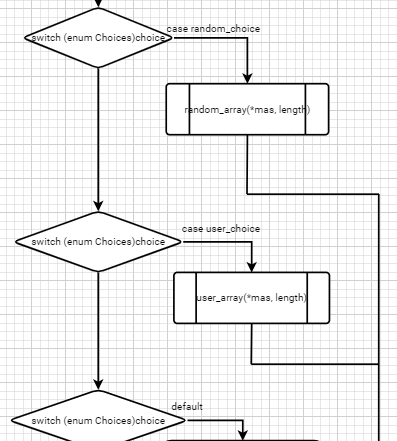


Рисунок ­ Блок-схема основного алгоритма часть 2

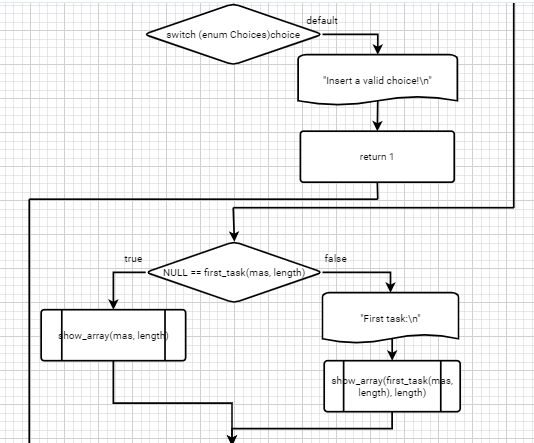


Рисунок  ­ Блок-схема основного алгоритма часть 3

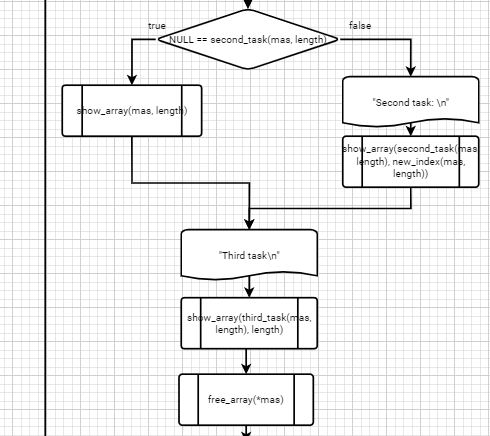


Рисунок  – Блок-схема основного алгоритма часть 4

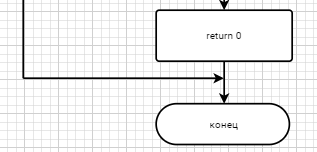


Рисунок  – Блок-схема основного алгоритма часть 5

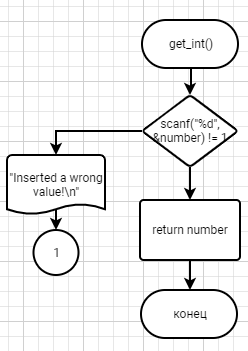


Рисунок  – Блок-схема функции get\_int()

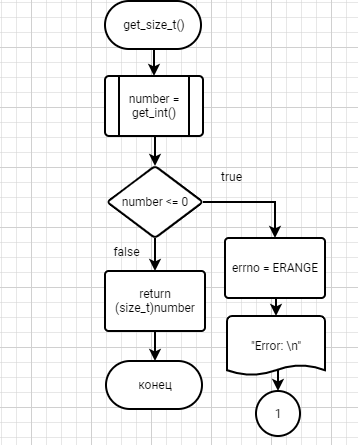


Рисунок  – Блок-схема функции get\_size\_t()

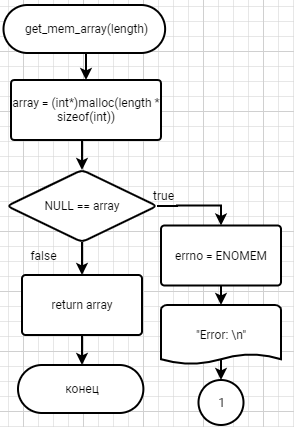


Рисунок 32 – Блок-схема функции get\_mem\_array(length)

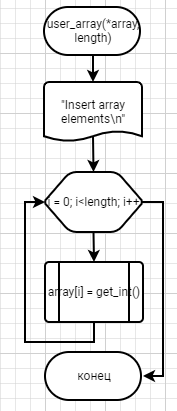


Рисунок 33 – Блок-схема функции user\_array(\*array, length)

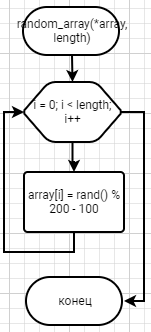


Рисунок 34 – Блок-схема функции random\_array(\*array, length)

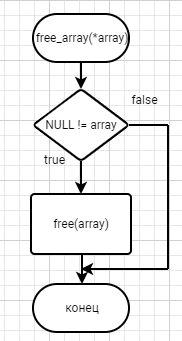


Рисунок  – Блок-схема функции free\_array(array)

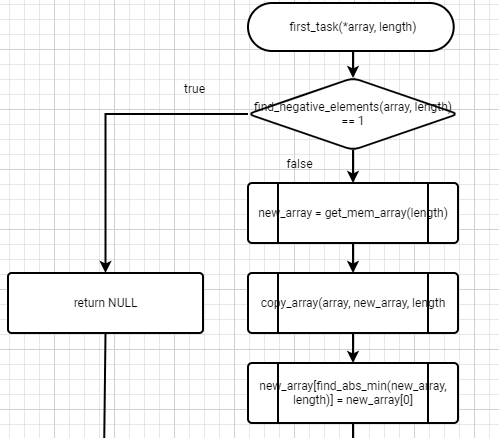


Рисунок 36 –Блок-схема функции first\_task(\*array, length) часть 1

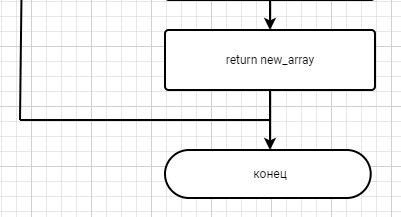


Рисунок  – Блок-схема функции first\_task(\*array, length) часть 2

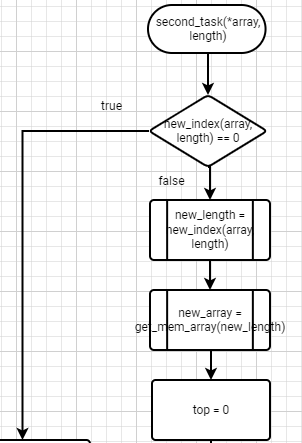


Рисунок 38 –Блок-схема функции second\_task(\*array, length) часть 1

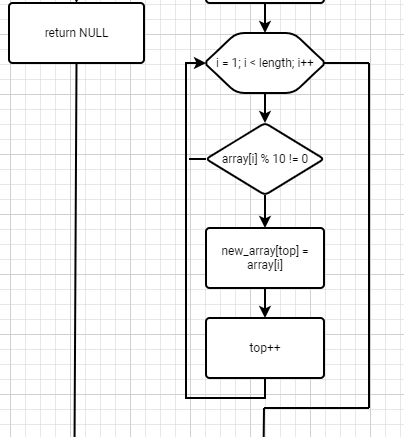


Рисунок  – Блок-схема функции second\_task(\*array, length) часть 2

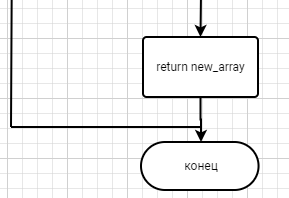


Рисунок  – Блок-схема функции second\_task(\*array, length) часть 3

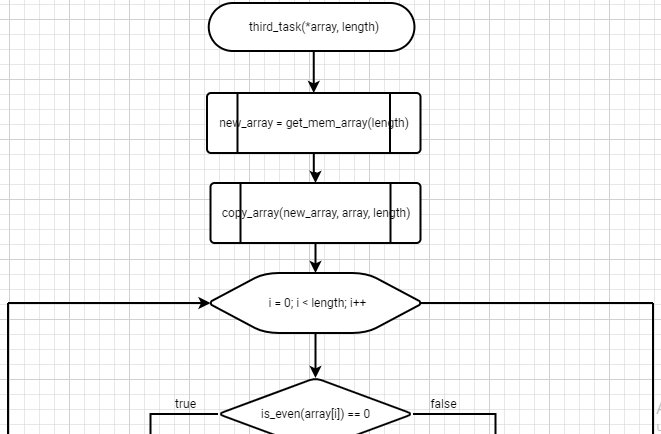


Рисунок 41 – Блок-схема функции third\_task(\*array, length) часть 1

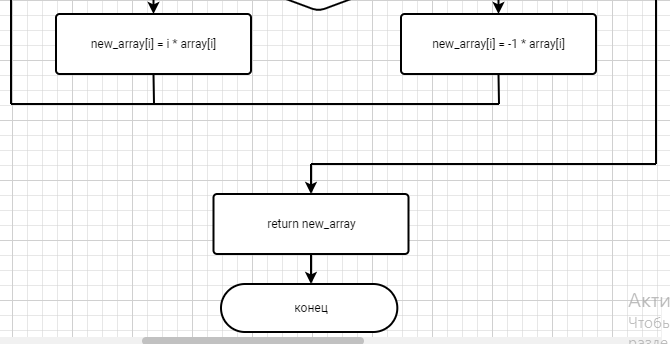


Рисунок  – Блок-схема функции third\_task(\*array, length) часть 2

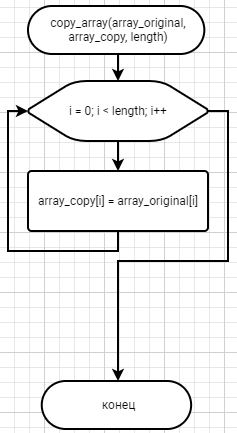


Рисунок 43 – Блок-схема функции copy\_array(array original, array\_copy, length)

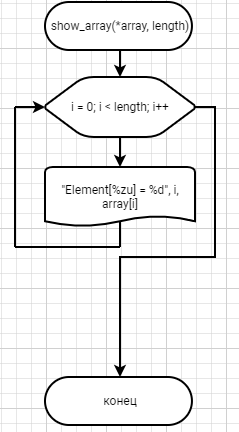


Рисунок 44 – Блок-схема функции show\_array(\*array, length)

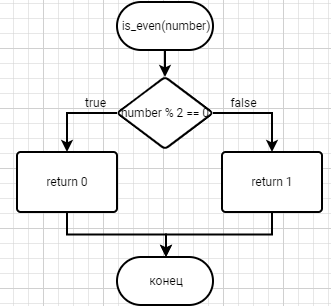


Рисунок  – Блок-схема функции is\_even(number)

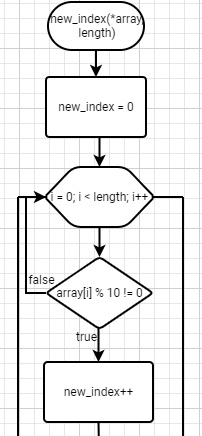


Рисунок 46 – Блок-схема функции new\_index(\*array, length) часть 1

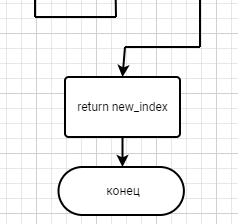


Рисунок  – Блок-схема функции new\_index(\*array, length) часть 2

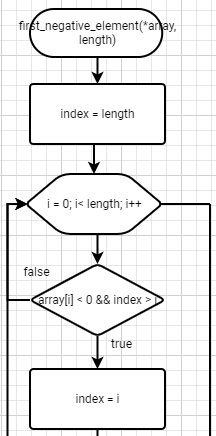


Рисунок 48 – Блок-схема функции first\_negative\_element(\*array, length) часть 1

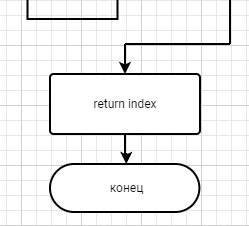


Рисунок 49 – Блок-схема функции first\_negative\_element(\*array, length) часть 2

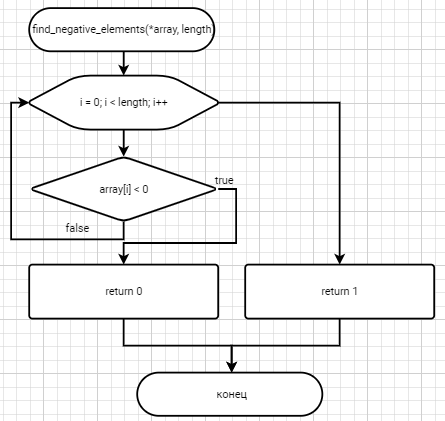


Рисунок 50 – Блок-схема функции find\_negative\_elements(\*array, length)

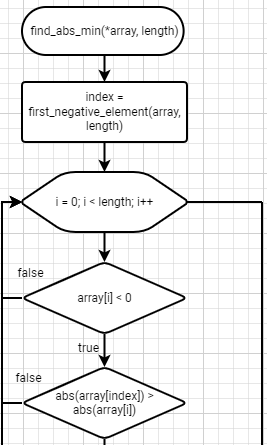


Рисунок 51 – Блок-схема функции find\_abs\_min(\*array, length) часть 1

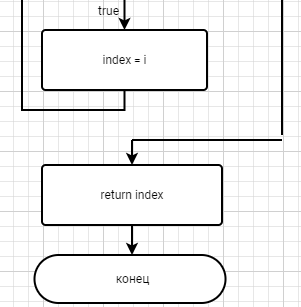


Рисунок  – Блок-схема функции find\_abs\_min(\*array, length) часть 2

* 1. Текст программы на языке C

#include <stdio.h>

#include <stdlib.h>

#include <malloc.h>

#include <errno.h>

#include <math.h>

/\*\*

\* @brief Функция присваивает целочисленное значение переменной

\* @return целочисленная цифра

\*/

int get\_int();

/\*\*

\* @brief Функция присваивает переменной целочисленное значение и проверяет его на положителность

\* @return number положительное число

\*/

size\_t get\_size\_t();

/\*\*

\* @brief структура хранит константы, указывающие выбор пользователя заполнить массив

\* @brief user\_choice хранит значение, вызывающее ввод массива вручную

\* @brief random\_choice хранит значение, вызывающее ввод массива случайными числами

\*/

enum Choices

{

user\_choice = 1,

random\_choice = 2,

};

/\*\*

\* @brief Функция выделяет память под массив

\* @param length длина массива

\* @return array указатель на пустой массив

\*/

int\* get\_mem\_array(const int length);

/\*\*

\* @brief Функция заполняет массив, элементы вводит пользователь

\* @param array указатель на заполняемый массив

\* @param length длина массива

\*/

void user\_array(int\* const array, const size\_t length);

/\*\*

\* @brief Функция заполняет массив рандомными числами в диапазоне [-100:200]

\* @param array указатель на заполняемый массив

\* @param length длина массива

\*/

void random\_array(int\* const array, const size\_t length);

/\*\*

\* @brief Функция, показывающая массив поэлементно

\* @param array ссылка на показываемый массив

\* @param length длина массива

\*/

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

/\*\*

\* @brief Функция копирующая элементы одного массива в другой

\* @param array\_original исходный массив

\* @param array\_copy пустой массив

\* @param length длина массива

\*/

void copy\_array(int\* const array\_original, int\* array\_copy, const size\_t length);

/\*\*

\* @brief Функция проверяет массив на предмет отрицательного числа

\* @param array ссылка на массив

\* @param length длина массива

\* @return 0 Есть такие элементы

\* @return 1 Нет таких элементов

\*/

int find\_negative\_elements(int\* const array, size\_t length);

/\*\*

\* @brief Функция находит индекс первого отрицательного элемента

\* @param array ссылка на массив

\* @param length длина массива

\* @return index индекс элемента

\*/

int first\_negative\_element(int\* const array, size\_t length);

/\*\*

\* @brief Функция находит по индексу минимальное по модулю отрицателное число

\* @param array ссылка на массив

\* @param length длина массива

\* @return index индекс элемента

\*/

int find\_abs\_min(int\* const array, size\_t length);

/\*\*

\* @brief Заменяет минимальное по модулю отрицателное число на первый элемент

\* @param array ссылка на массив

\* @param length размер массива

\* @return NULL отрицательных чисел нет

\* @return new\_array новый массив, соответствующий этим требованиям

\*/

int \*first\_task(int\* array, size\_t length);

/\*\*

\* @brief Функция вычисляет размерность нового массива по условию из 2 пункта

\* @param array ссылка на массив

\* @param length длина массива

\* @return new\_index размерность нового массива

\*/

int new\_index(int\* array, size\_t length);

/\*\*

\* @brief Функция удаляет из массива элементы, оканчивающиеся на 0

\* @param array ссылка на массив

\* @param length длина массива

\* @return NULL в массиве нет элементов, окачивающиеся на 0

\* @return new\_array новый массив, соответствующий этим требованиям

\*/

int\* second\_task(int\* const array, size\_t length);

/\*\*

\* @brief Функция проверяет на четность целое число

\* @param number число

\* @return 0 четный

\* @return 1 нечетный

\*/

int is\_even(int number);

/\*\*

\* @brief Заменяет в массиве элементы по соответствующим условиям

\* @param array ссылка на массив

\* @param length размерность массива

\* @return new\_array новый массив, соответствующий этим требованиям

\*/

int\* third\_task(int\* const array, size\_t length);

/\*\*

\* @brief Функция, освобождающая массив

\* @param array указатель на массив

\*/

void free\_array(int\* array);

/\*\*

\* @brief Точка входа в программу

\* @return 0 Программа исправна

\*/

int main()

{

puts("insert a length of array\n");

size\_t length = get\_size\_t();

puts("if you fill array by youself, press 1, if you fill array by random numbers, press 2\n");

int choice = get\_int();

int\* mas = get\_mem\_array(length);

switch ((enum Choices)choice)

{

case random\_choice:

random\_array(\*mas, length);

break;

case user\_choice:

user\_array(\*mas, length);

break;

default:

puts("Insert a valid choice!\n");

return 1;

}

if (NULL == first\_task(\*mas, length))

{

show\_array(\*mas, length);

}

else:

{

puts("First task:\n");

show\_array(first\_task(\*mas, length), length);

}

if (NULL == second\_task(\*mas, length))

{

show\_array(\*mas, length);

}

else

{

show\_array(second\_task(\*mas, length), new\_index(\*mas, length));

}

show\_array(third\_task(\*mas, length), length);

free\_array(\*mas);

return 0;

}

int get\_int()

{

int number;

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

{

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

abort();

}

return number;

}

size\_t get\_size\_t()

{

int number = get\_int();

if (number <= 0)

{

errno = ERANGE;

perror("Error: \n");

abort();

}

return (size\_t)number;

}

int\* get\_mem\_array(const int length)

{

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

if (NULL == array)

{

errno = ENOMEM;

perror("Error: ");

abort();

}

return array;

}

void user\_array(int\* const array, const size\_t length)

{

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

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

{

array[i] = get\_int();

}

}

void random\_array(int\* const array, const size\_t length)

{

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

{

array[i] = rand() % 200 - 100;

}

}

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

{

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

{

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

}

}

void copy\_array(int\* const array\_original, int\* array\_copy, const size\_t length)

{

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

{

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

}

}

int find\_negative\_elements(int\* const array, size\_t length)

{

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

{

if (array[i] < 0)

{

return 0;

}

}

return 1;

}

int first\_negative\_element(int\* const array, size\_t length)

{

int index = length;

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

{

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

{

index = i;

}

}

return index;

}

int find\_abs\_min(int\* const array, size\_t length)

{

int index = first\_negative\_element(\*array, length);

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

{

if (array[i] < 0)

{

if (abs(array[index]) > abs(array[i]))

{

index = i;

}

}

}

return index;

}

int new\_index(int\* array, size\_t length)

{

int new\_index = 0;

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

{

if (array[i] % 10 != 0)

{

new\_index++;

}

}

return new\_index;

}

int is\_even(int number)

{

if (number % 2 == 0)

{

return 0;

}

return 1;

}

int \*first\_task(int\* array, size\_t length)

{

if (find\_negative\_elements(array, length) == 1)

{

return NULL;

}

int\* new\_array = get\_mem\_array(length);

copy\_array(\*array, \*new\_array, length);

new\_array[find\_abs\_min(\*new\_array, length)] = new\_array[0];

return new\_array;

}

int\* second\_task(int\* const array, size\_t length)

{

if (new\_index(\*array, length) == 0)

{

return NULL;

}

int new\_length = new\_index(\*array, length);

int\* new\_array = get\_mem\_array(new\_length);

int top = 0;

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

{

if (array[i] % 10 != 0)

{

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

top++;

}

}

return new\_array;

}

int\* third\_task(int\* const array, size\_t length)

{

int\* new\_array = get\_mem\_array(length);

copy\_array(\*array, \*new\_array, length);

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

{

if (is\_even(array[i]) == 0)

{

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

}

else

{

new\_array[i] = -1 \* array[i];

}

}

return new\_array;

}

void free\_array(int\* array)

{

if (NULL != array)

{

free(array);

}

}

* 1. Результаты выполнения программы

Результаты выполнения программы представлены ниже (Рисунок 58).

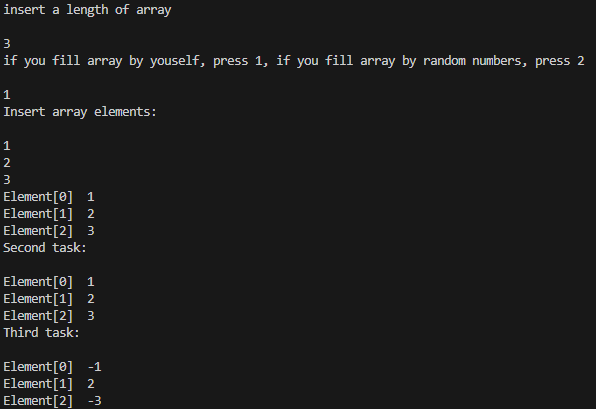


Рисунок  – Результат выполнения программы

* 1. Выполнение тестовых примеров

В программе MS Excel выполнены тестовые примеры. Результаты их выполнения представлены ниже (Рисунок 11, Рисунок 12).

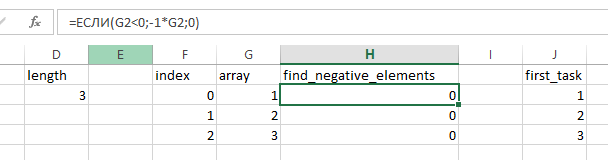


Рисунок 54 – Результат расчета функции find\_negative\_elements(\*array, length)

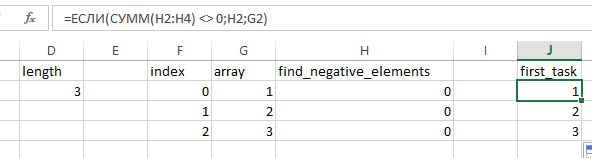


Рисунок 55 – Результат расчета функции first\_task(\*array, length)

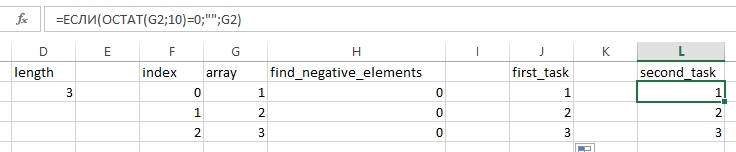


Рисунок  – Результат расчета функции second\_task(\*array, length)

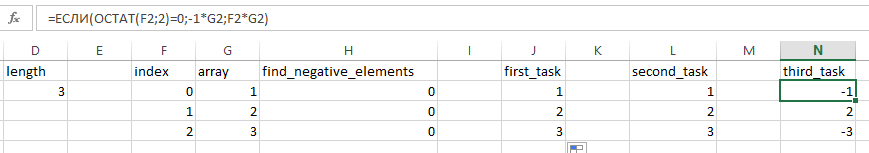
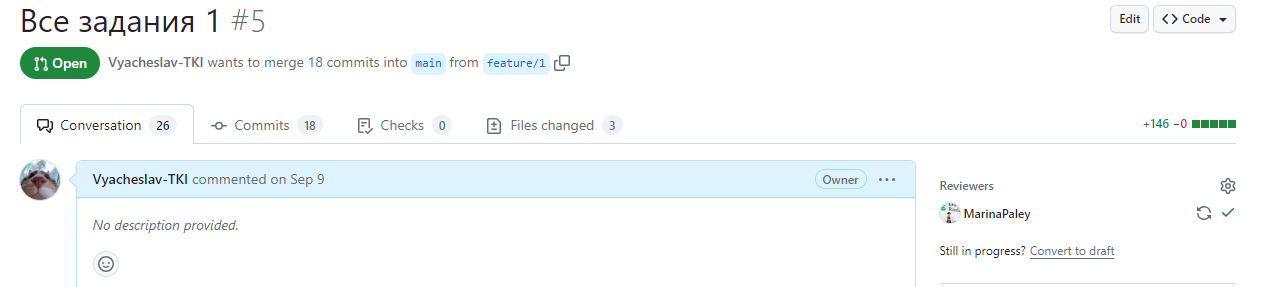


Рисунок  – Результат расчета функции third\_task(\*array, length)

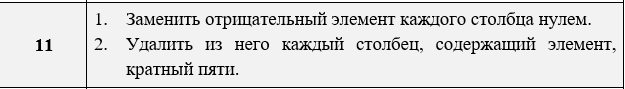
* 1. Отметка о выполнении задания в веб-хостинге системы контроля версий



1. 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. Данные указаны в таблице (Таблица 3).

Таблица  – Исходные данные



* 1. Блок-схема алгоритма

Блок-схема основного алгоритма представлена ниже (Рисунок 58, Рисунок 59, Рисунок 60, Рисунок 61, Рисунок 62, Рисунок 63). Блок-схемы вспомогательных функций представлены ниже (Рисунок 64, Рисунок 65, Рисунок 66, Рисунок 67, Рисунок 68, Рисунок 69, Рисунок 70, Рисунок 71, Рисунок 72, Рисунок 73, Рисунок 74, Рисунок 75, Рисунок 76, Рисунок 77, Рисунок 78, Рисунок 79, Рисунок 80, Рисунок 81, Рисунок 82, Рисунок 83, Рисунок 84, Рисунок 85).

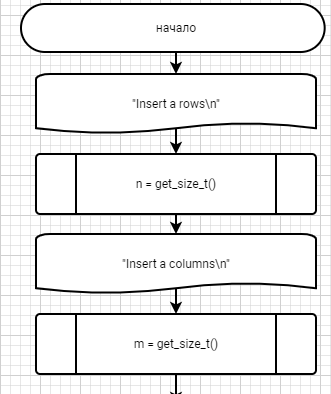


Рисунок  ­ Блок-схема основного алгоритма часть 1

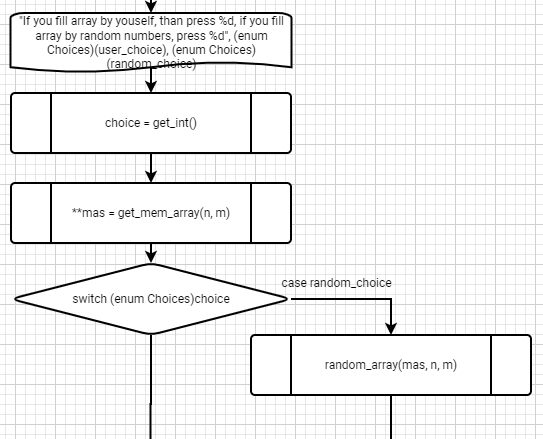


Рисунок  – Блок-схема основного алгоритма часть 2

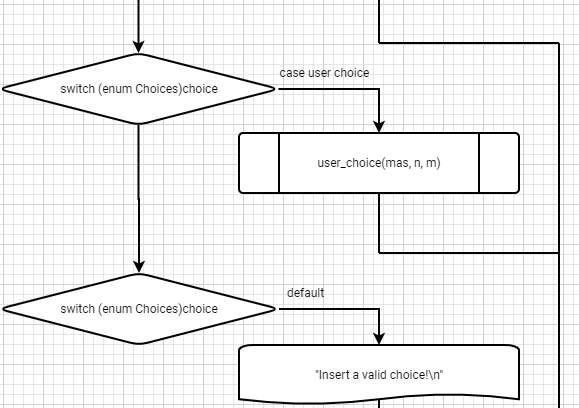


Рисунок  – Блок-схема основного алгоритма часть 3

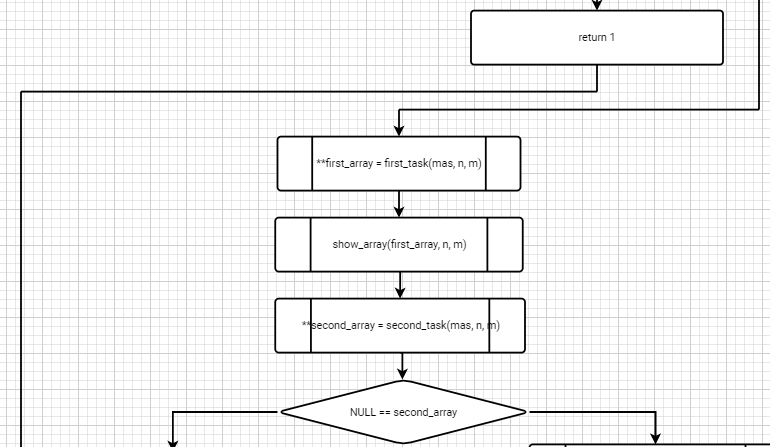


Рисунок  – Блок-схема основного алгоритма часть 4

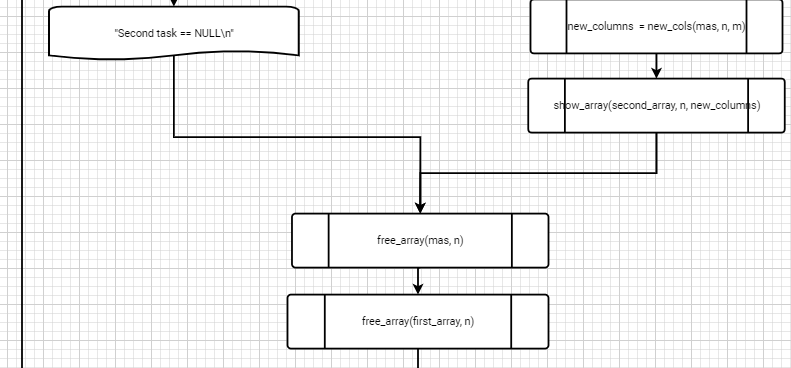


Рисунок  – Блок-схема основного алгоритма часть 5

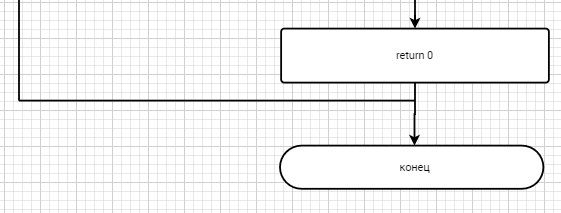


Рисунок  – Блок-схема основного алгоритма часть 6

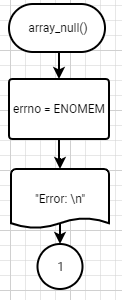


Рисунок  – Блок-схема функции array\_null()

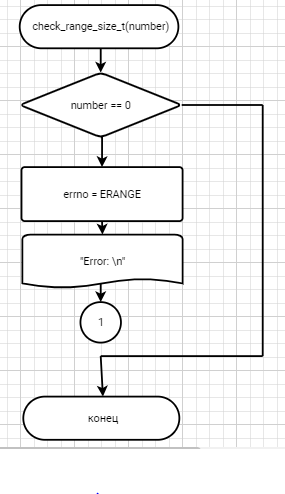


Рисунок 65 – Блок-схема функции check\_range\_size\_t(number)

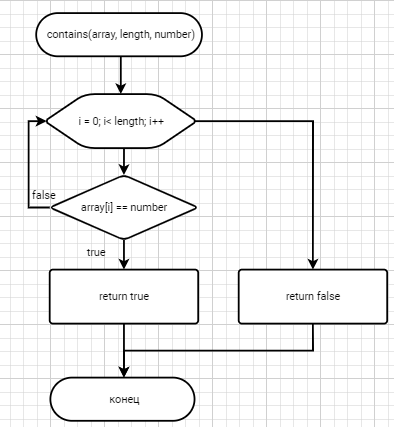


Рисунок 66 – Блок-схема функции contains(array, length, number)

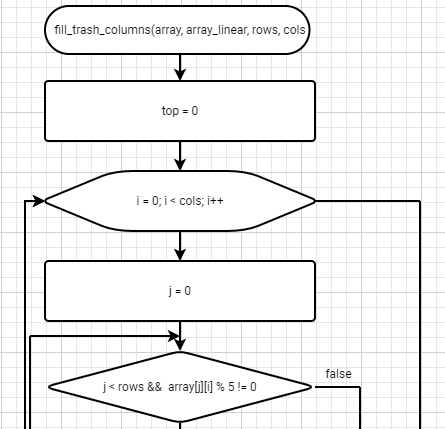


Рисунок 67 – Блок-схема функции fill\_trash\_columns(array, array\_linear, rows, cols) часть 1

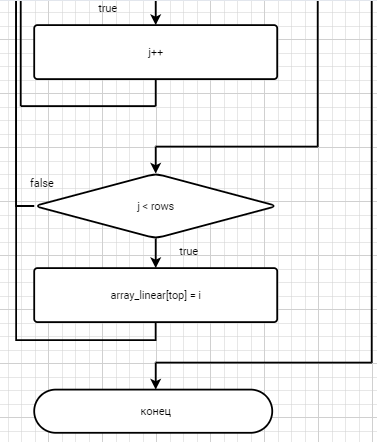


Рисунок 68 – Блок-схема функции fill\_trash\_columns(array, array\_linear, rows, cols) часть 2

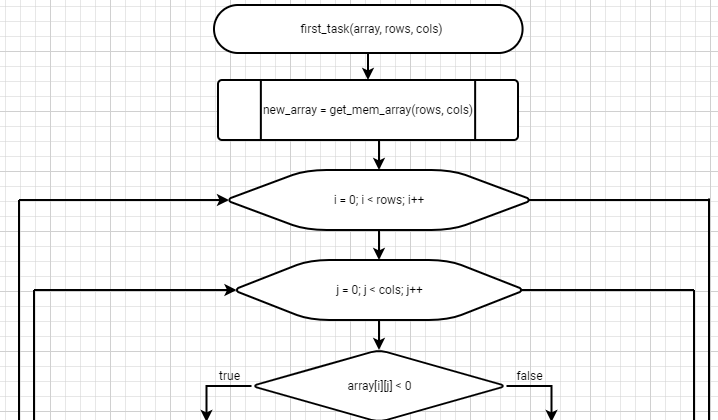


Рисунок  – Блок-схема функции first\_task(array, rows, cols) часть 1

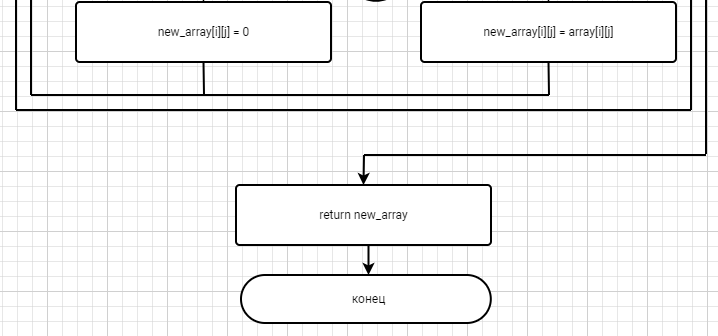


Рисунок  – Блок-схема функции first\_task(array, rows, cols) часть 2

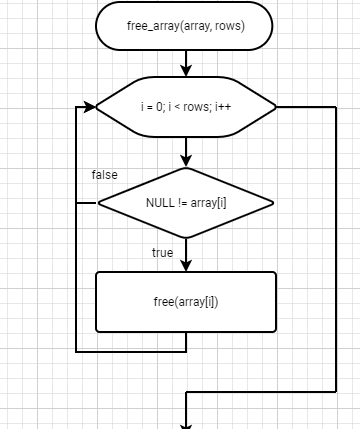


Рисунок 71 – Блок схема функции free\_array(array, rows) часть 1

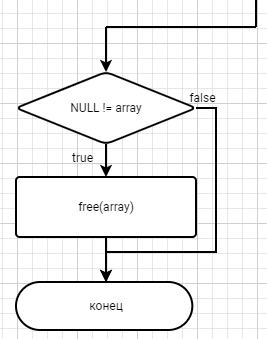


Рисунок  – Блок-схема функции free\_array(array, rows) часть 2

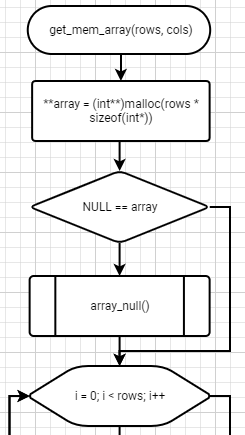


Рисунок 73 – Блок-схема функции get\_mem\_array(rows, cols) часть 1

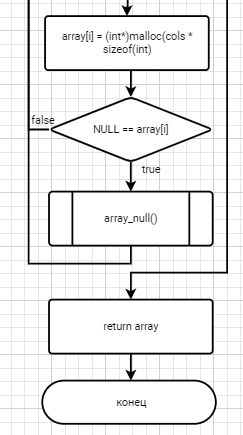


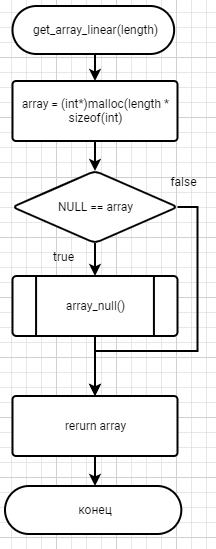
Рисунок  – Блок-схема функции get\_mem\_array(rows, cols) часть 2

Рисунок 75 – Блок-схема функции get\_array\_linear(length)

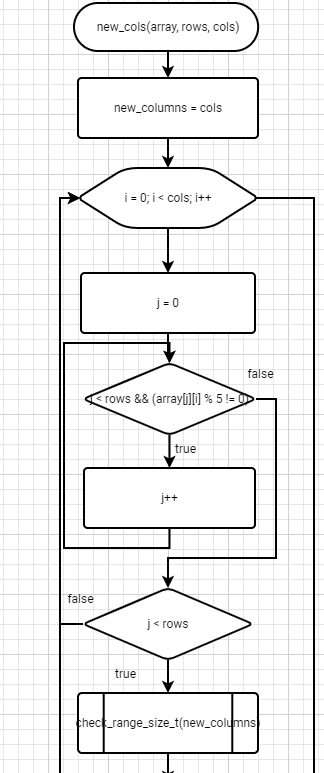


Рисунок 76 – Блок-схема функции new\_cols(array, rows, cols) часть 1

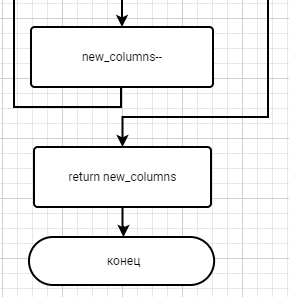


Рисунок  – Блок-схема функции new\_cols(array, rows, cols) часть 2

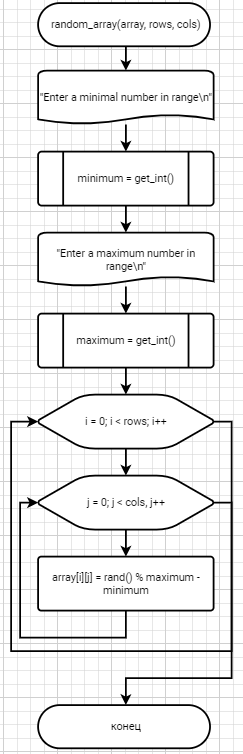


Рисунок 78 – Блок-схема функции random\_array(array, rows, cols)

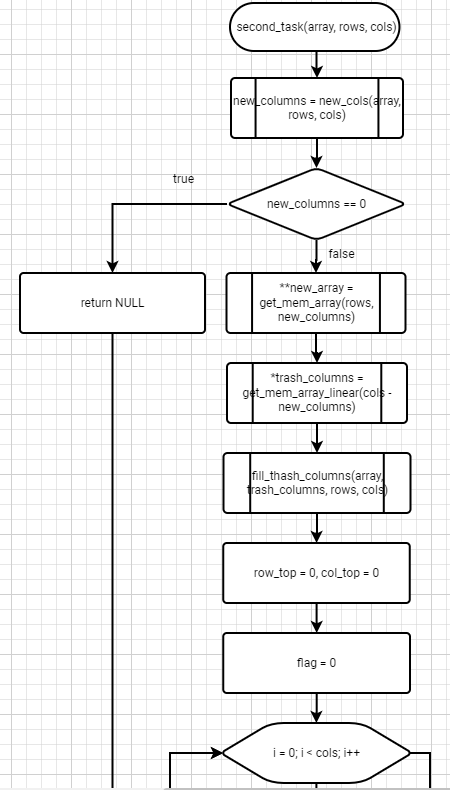


Рисунок 79 – Блок-схема функции second\_task(array, rows, cols) часть 1

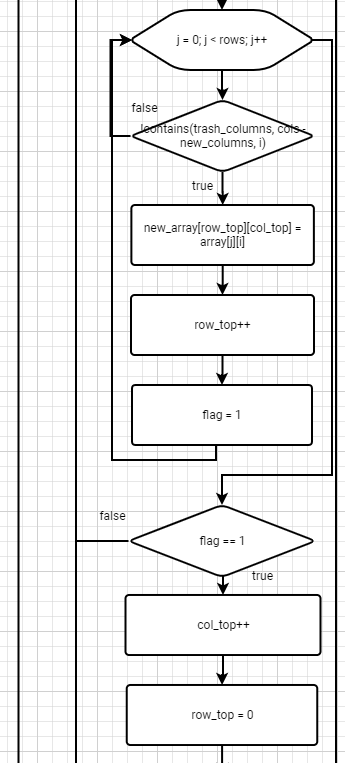


Рисунок  – Блок-схема функции second\_task(array, rows, cols) часть 2

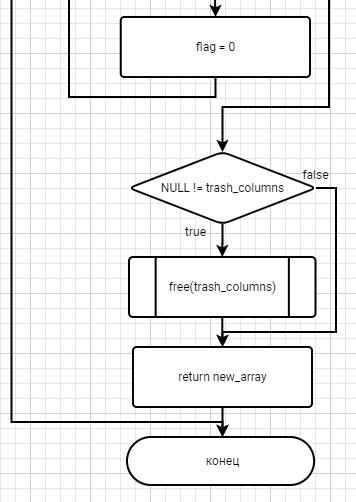


Рисунок  – Блок-схема функции second\_task(array, rows, cols) часть 3

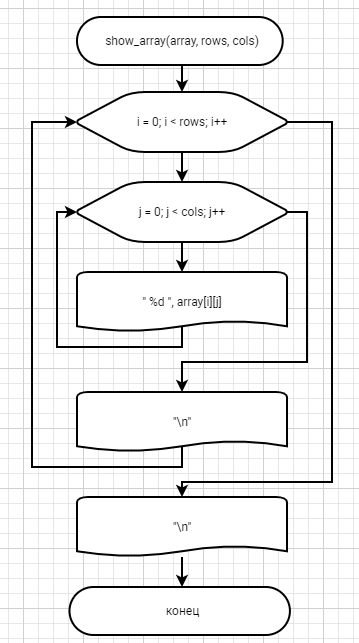


Рисунок  Блок-схема функции show\_array(array, rows, cols)

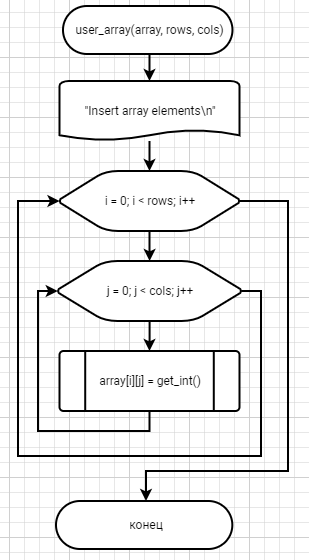


Рисунок  – Блок-схема функции user\_array(array, rows, cols)

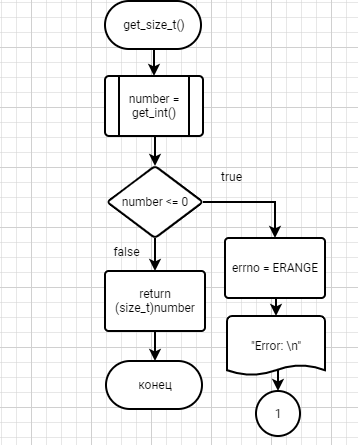


Рисунок  – Блок-схема функции get\_size\_t()

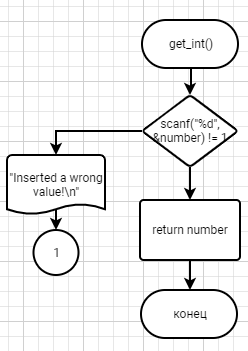


Рисунок  – Блок-схема функции get\_int()

* 1. Текст программы на языке C

#include <stdio.h>

#include <stdlib.h>

#include <malloc.h>

#include <errno.h>

#include <stdbool.h>

/\*\*

\* @brief Функция присваивает целочисленное значение переменной

\* @return целое число

\*/

int get\_int();

/\*\*

\* @brief Функция, прерывающая программу из-за отсутствия памяти для массивов

\*/

void array\_null();

/\*\*

\* @brief освобождает многомерный массив

\* @param array указатель на заполняемый массив

\* @param rows количество строк в массиве

\*/

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

/\*\*

\* @brief Функция присваивает переменной целочисленное значение и проверяет его на положителность

\* @return number положительное число

\*/

size\_t get\_size\_t();

/\*\*

\* @brief структура хранит константы, указывающие выбор пользователя заполнить массив

\* @brief user\_choice хранит значение, вызывающее ввод массива вручную

\* @brief random\_choice хранит значение, вызывающее ввод массива случайными числами

\*/

enum Choices

{

user\_choice = 1,

random\_choice = 2,

};

/\*\*

\* @brief Функция выделяет память под многомерный массив

\* @param rows количество строк в массиве

\* @param cols количество столбцов в массиве

\* @return array указатель на пустой многомерный массив

\*/

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

/\*\*

\* @brief Функция заполняет массив, элементы вводит пользователь

\* @param array указатель на заполняемый массив

\* @param rows количество строк в массиве

\* @param cols количество столбцов в массиве

\*/

void user\_array(int\*\* const array, const size\_t rows, const size\_t cols);

/\*\*

\* @brief Функция заполняет массив рандомными числами в диапазоне, указанный пользователем

\* @param array указатель на заполняемый массив

\* @param rows количество строк в массиве

\* @param cols количество столбцов в массиве

\*/

void random\_array(int\*\* const array, const size\_t rows, const size\_t cols);

/\*\*

\* @brief Функция, показывающая массив поэлементно

\* @param array ссылка на показываемый массив

\* @param rows количество строк в массиве

\* @param cols количество столбцов в массиве

\*/

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

/\*\*

\* @brief Заменяет отрицателный элемент каждого столбца нулем

\* @param array ссылка на массив

\* @param rows количество строк в массиве

\* @param cols количество столбцов в массиве

\* @return NULL отрицательных чисел нет

\* @return new\_array новый массив, соответствующий этим требованиям

\*/

int \*\*first\_task(int\*\* array, size\_t rows, size\_t cols);

/\*\*

\* @brief Функция удаляет из массива столбцы, содержащие элемент, кратный 5

\* @param array ссылка на массив

\* @param rows количество строк в массиве

\* @param cols количество столбцов в массиве

\* @return result\_array новый массив, соответствующий этим требованиям

\*/

int \*\*second\_task(int\*\* const array, size\_t rows, size\_t cols);

/\*\*

\* @brief Функция вычисляет размер столбцов в массиве, который соответствует условию 2-го пункта

\* @param array ссылка на массив

\* @param rows количество строк в массиве

\* @param cols количество столбцов в массиве

\* @return new\_columns количество столбцов нового массива

\*/

size\_t new\_cols(const int\*\* const array, size\_t rows, size\_t cols);

/\*\*

\* @brief Вспомогательная функция для функции new\_cols, предотвращает выход за диапазон переменной

\* @param number положительное число (или нуль)

\*/

void check\_range\_size\_t(size\_t number);

/\*\*

\* @brief выделяет память под линейный массив

\* @param length длина массива

\* @return array пустой массив

\*/

int\* get\_mem\_array\_linear(const size\_t length);

/\*\*

\* @brief Функция заполняет вспомогательный массив ненужных столбцов

\* @param array многомерный массив

\* @param array\_linear массив, который будет заполняться

\* @param rows количество строк в массиве

\* @param cols количество столбцов в массиве

\*/

void fill\_trash\_columns(const int\*\* const array, int\* array\_linear, size\_t rows, size\_t cols);

/\*\*

\* @brief Функция, проверяющая, что есть ли число в массиве

\* @param array линейный массив

\* @param length длина массива

\* @param number проверяемое число

\* @return true есть число

\* @return false нет числа

\*/

bool contains(const int\* const array, const size\_t length, const size\_t number);

/\*\*

\* @brief точка входа в программу

\* @return 0 программа исправна

\* @return 1 программа завершилась досрочно с ошибкой

\*/

int main()

{

puts("insert a rows\n");

size\_t n = get\_size\_t();

puts("insert a columns\n");

size\_t m = get\_size\_t();

printf("if you fill array by youself, press %d, if you fill array by random numbers, press %d\n", (enum Choices)(user\_choice), (enum Choices)(random\_choice));

int choice = get\_int();

int\*\* mas = get\_mem\_array(n, m);

switch ((enum Choices)choice)

{

case random\_choice:

random\_array(mas, n, m);

break;

case user\_choice:

user\_array(mas, n, m);

break;

default:

puts("Insert a valid choice!\n");

return 1;

}

show\_array(mas, n, m);

int\*\* first\_array = first\_task(mas, n, m);

show\_array(first\_array, n, m);

int\*\* second\_array = second\_task(mas, n, m);

if (NULL == second\_array)

{

puts("Second task == NULL\n");

}

else

{

size\_t new\_columns = new\_cols(mas, n, m);

show\_array(second\_array, n, new\_columns);

}

free\_array(mas, n);

free\_array(first\_array, n);

free\_array(second\_array, n);

return 0;

}

int get\_int()

{

int number;

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

{

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

abort();

}

return number;

}

size\_t get\_size\_t()

{

int number = get\_int();

if (number <= 0)

{

errno = ERANGE;

perror("Error: \n");

abort();

}

return (size\_t)number;

}

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

{

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

if (NULL == array)

{

array\_null();

}

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

{

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

if (NULL == array[i])

{

array\_null();

}

}

return array;

}

void user\_array(int\*\* const array, const size\_t rows, const size\_t cols)

{

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

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

{

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

{

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

}

}

}

void random\_array(int\*\* const array, const size\_t rows, const size\_t cols)

{

puts("Enter a minimal number in range\n");

int minimum = get\_int();

puts("Enter a maximum number in range\n");

int maximum = get\_int();

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

{

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

{

array[i][j] = rand() % maximum - minimum;

}

}

}

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

{

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

{

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

{

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

}

printf("\n");

}

puts("\n");

}

int \*\*first\_task(int\*\* array, size\_t rows, size\_t cols)

{

int\*\* new\_array = get\_mem\_array(rows, cols);

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

{

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

{

if (array[i][j] < 0)

{

new\_array[i][j] = 0;

}

else

{

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

}

}

}

return new\_array;

}

void check\_range\_size\_t(size\_t number)

{

if (number == 0)

{

errno = ERANGE;

perror("Error: ");

abort();

}

}

size\_t new\_cols(const int\*\* const array, size\_t rows, size\_t cols)

{

size\_t new\_columns = cols;

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

{

size\_t j = 0;

while (j < rows && (array[j][i] % 5 != 0))

{

j++;

}

if (j < rows)

{

check\_range\_size\_t(new\_columns); // Проверяется на выход из диапазона!

new\_columns--;

}

}

return new\_columns;

}

int\* get\_mem\_array\_linear(const size\_t length)

{

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

if (NULL == array)

{

array\_null();

}

return array;

}

void fill\_trash\_columns(const int\*\* const array, int\* array\_linear, size\_t rows, size\_t cols)

{

size\_t top = 0;

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

{

size\_t j = 0;

while (j < rows && (array[j][i] % 5 != 0))

{

j++;

}

if (j < rows)

{

array\_linear[top] = i;

}

}

}

bool contains(const int\* const array, const size\_t length, const size\_t number)

{

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

{

if (number == array[i])

{

return true;

}

}

return false;

}

int \*\*second\_task(int\*\* const array, size\_t rows, size\_t cols)

{

size\_t new\_columns = new\_cols(array, rows, cols);

if (new\_columns == 0)

{

return NULL;

}

int\*\* new\_array = get\_mem\_array(rows, new\_columns);

int\* trash\_columns = get\_mem\_array\_linear(cols - new\_columns);

fill\_trash\_columns(array, trash\_columns, rows, cols);

size\_t row\_top = 0, col\_top = 0;

int flag = 0;

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

{

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

{

if (!contains(trash\_columns, (cols - new\_columns), i))

{

new\_array[row\_top][col\_top] = array[j][i];

row\_top++;

flag = 1;

}

}

if (flag == 1)

{

col\_top++;

row\_top = 0;

flag = 0;

}

}

if (NULL != trash\_columns)

{

free(trash\_columns);

}

return new\_array;

}

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

{

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

{

if (NULL != array[i])

{

free(array[i]);

}

}

if (NULL != array)

{

free(array);

}

}

void array\_null()

{

errno = ENOMEM;

perror("Error: ");

abort();

}}

* 1. Результаты выполнения программы

Результаты выполнения программы представлены ниже (Рисунок 86).

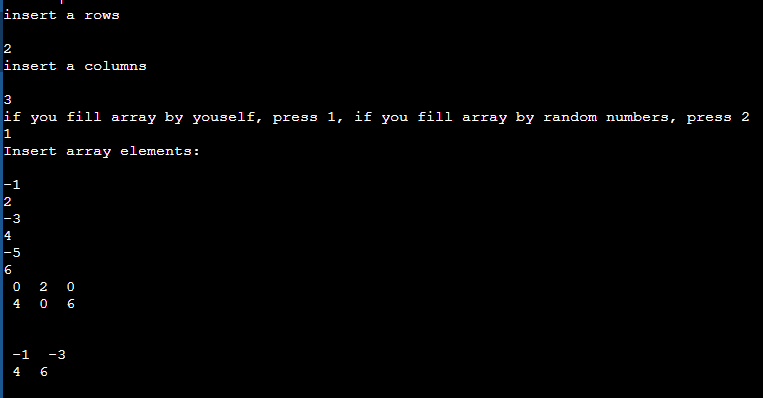


Рисунок  – Результат выполнения программы

* 1. Выполнение тестовых примеров

В программе MS Excel выполнены тестовые примеры. Результаты их выполнения представлены ниже (Рисунок 87. Рисунок 88).

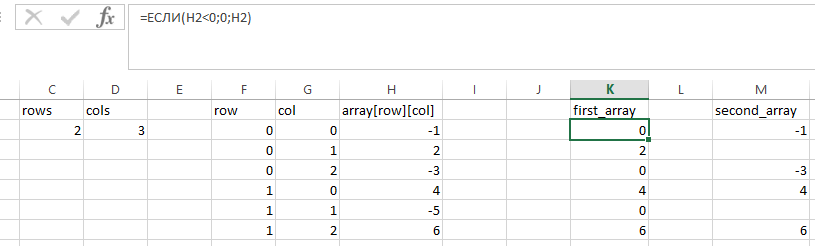


Рисунок  – Результат расчета функции first\_task(array, rows, cols)

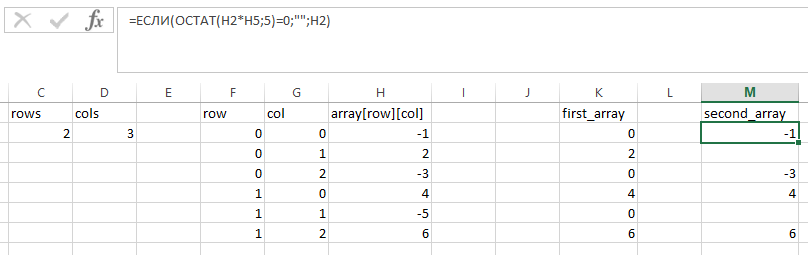


Рисунок  – Результат расчета функции second\_task(array, rows, cols)

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

