ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ  
«РОССИЙСКИЙ УНИВЕРСИТЕТ ТРАНСПОРТА»  
(РУТ (МИИТ))

Институт транспортной техники и систем управления

Кафедра «Управление и защита информации»

ОТЧЁТ  
О ЛАБОРАТОРНОЙ РАБОТЕ №4

По дисциплине «Языки программирования»

ВАРИАНТ 6

Выполнил: ст. гр. ТКИ-141

Винтфельд Рина Дмитриевна

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

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

Москва 2023

Оглавление

[1 Задача 4.1 3](#_Toc153831158)

[1.1 Формулировка задания 4.1 3](#_Toc153831159)

[1.2 Блок-схема алгоритма 4.1 4](#_Toc153831160)

[1.3 Текст программы на языке C 4.1 14](#_Toc153831161)

[1.4 Результаты выполнения программы 4.1 24](#_Toc153831162)

[1.5 Отметка о выполнении задания в веб-хостинге системы контроля версий 4.1 26](#_Toc153831163)

[2 Задача 4.2 27](#_Toc153831164)

[2.1 Формулировка задания 4.2 27](#_Toc153831165)

[2.2 Блок-схема алгоритма 4.2 28](#_Toc153831166)

[2.3 Текст программы на языке C 4.2 29](#_Toc153831167)

[2.4 Результаты выполнения программы 4.2 42](#_Toc153831168)

[2.5 Отметка о выполнении задания в веб-хостинге системы контроля версий 4.2 43](#_Toc153831169)

[3 Задача 4.3 44](#_Toc153831170)

[3.1 Формулировка задания 4.3 44](#_Toc153831171)

[3.2 Блок-схема алгоритма 4.3 45](#_Toc153831172)

[3.3 Текст программы на языке C 4.3 49](#_Toc153831173)

[3.4 Результаты выполнения программы 4.3 60](#_Toc153831174)

[3.5 Отметка о выполнении задания в веб-хостинге системы контроля версий 4.3 61](#_Toc153831175)

1. Задача 4.1
   1. Формулировка задания 4.1

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 – Исходные данные

|  |  |  |
| --- | --- | --- |
| Вариант | Задачи | Интервал |
| 6 | 1. Найти сумму отрицательных элементов, значения которых кратно10. 2. Заменить первые k элементов массива на те же элементы в обратном порядке. 3. Определить, есть ли пара соседних элементов с произведением, равным заданному числу. | [-1000;1000] |

* 1. Блок-схема алгоритма 4.1

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

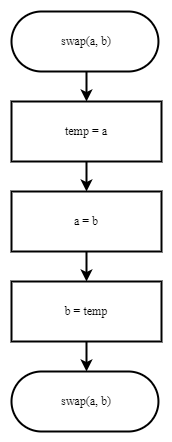
.

Рисунок 1 - Блок-схема используемой функции

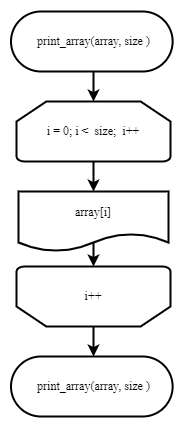


Рисунок 2 - Блок-схема используемой функции

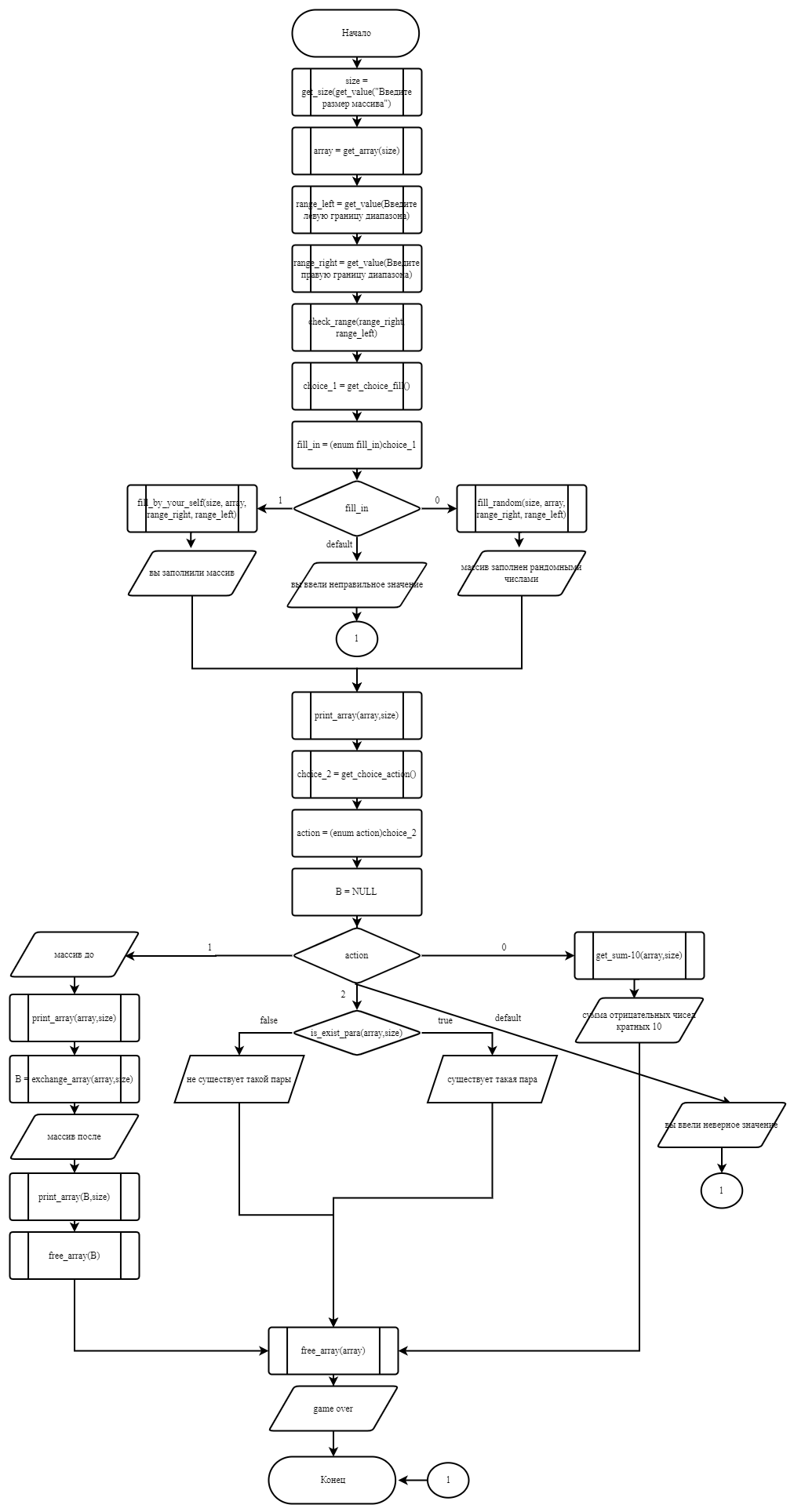


Рисунок - Блок-схема основного алгоритма

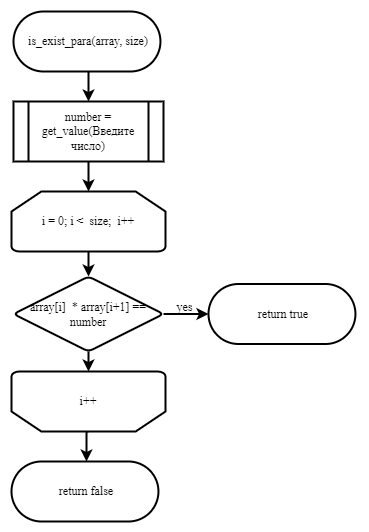


Рисунок - Блок-схема используемой функции



Рисунок - Блок-схема используемой функции

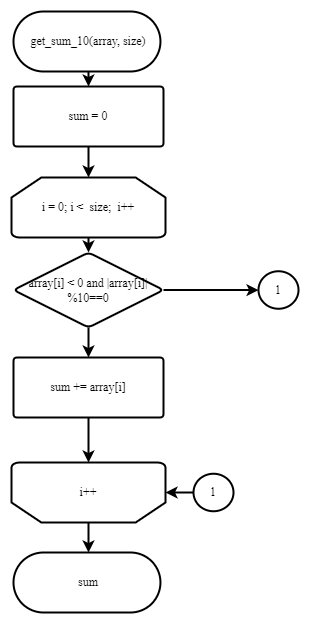


Рисунок - Блок-схема используемой функции



Рисунок - Блок-схема используемой функции



Рисунок - Блок-схема используемой функции



Рисунок - Блок-схема используемой функции

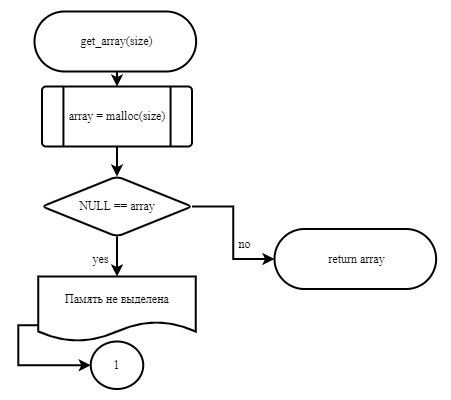


Рисунок - Блок-схема используемой функции

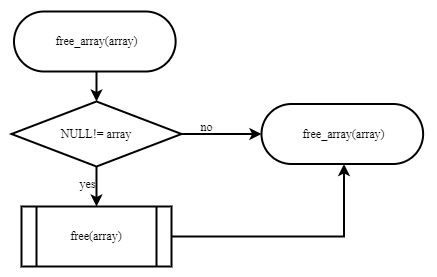


Рисунок - Блок-схема используемой функции

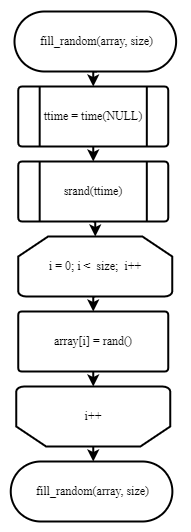


Рисунок - Блок-схема используемой функции

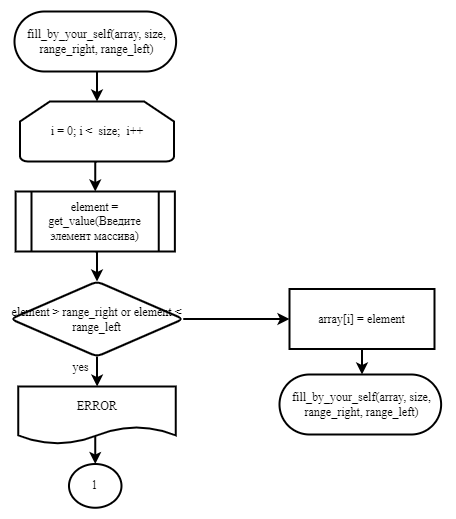


Рисунок - Блок-схема используемой функции

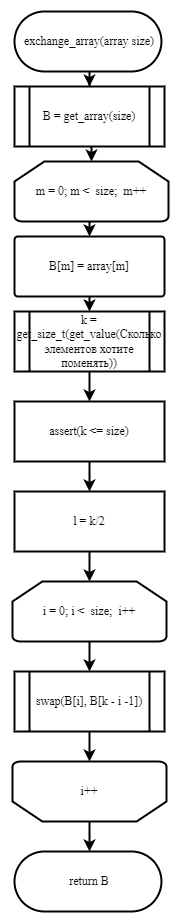


Рисунок - Блок-схема используемой функции

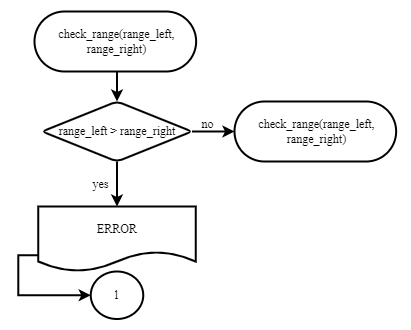


Рисунок - Блок-схема используемой функции

* 1. Текст программы на языке C 4.1

#include <stdio.h>

#include <stdlib.h>

#include <errno.h>

#include <time.h>

#include <stdbool.h>

#include <assert.h>

/\*

\* @brief Выбор пользователя

\* @brief fill\_random - Заполнение массива пользователем

\* @brief fill\_by\_my\_self - Заполнение массива рандомными числами

\*/

enum fill\_in{fill\_randomm, fill\_by\_my\_self};

/\*

\* @brief Выбор пользователя

\* @brief sum\_10 - Найти сумму отрицательных элементов, значения которых кратно 10

\* @brief exchange - Заменить первые k элементов массива на те же элементы в обратном порядке

\* @brief para - Определить, есть ли пара соседних элементов с произведением, pавным заданном числу

\*/

enum action {sum\_10, exchange, para};

/\*

\* @brief Считает сумму отрицательных чисел, кратных 10

\* @param array - массив

\* @param size - размер массива

\* @return sum - сумма отрицательных чисел, кратных 10

\*/

int get\_sum\_10(int\* array, size\_t size);

/\*

\* @brief Заняет первые k элементов на те же элементы, записанные в обратном порядке

\* @param array - массив

\* @param size - размер массива

\* @return изменённый массив

\*/

int\* exchange\_array(int\* array,size\_t size);

/\*

\* @brief меняет переменные

\* @param a - переменная

\* @param b - переменная

\*/

void swap(int\* a, int\* b);

/\*

\* @brief определяет существует ли такая пара элементов с произедением, равным заданному числу

\* @param array - массив

\* @param size - размер массива

\* @return True/False

\*/

bool is\_exist\_para(int\* array, size\_t size);

/\*

\* @brief предлагает выбор пользователю

\* @return введенный номер действия

\*/

int get\_choice\_fill();

/\*

\* @brief предлагает выбор пользователю

\* @return введенный номер действия

\*/

int get\_choice\_action();

/\*

\* @brief проверяет введенное значение

\* @param massage - сообщение для пользователя

\* @return переменную

\*/

int get\_value(const char\* massage);

/\*

\* @brief проверяет или введенное значение > 0

\* @param value введенное значение

\* @return размер массива

\*/

size\_t get\_size\_t(int value);

/\*

\* @brief проверяет или введенный диапазон соответствует действительности

\* @param range\_right - правая граница диапазона

\* @param range\_left - левая граница диапазона

\*/

void check\_range(int range\_right, int range\_left);

/\*

\* @brief создает массив

\* @param size - размер массива

\* @return массив

\*/

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

/\*

\* @brief Заполняет массив рандомными числами

\* @param size - размер массива

\* @param array - массив

\* @param range\_right - правая граница диапазона

\* @param range\_left - левая граница диапазона

\* @return 0 - в случае успеха

\*/

int fill\_random(const size\_t size, int\* array, int range\_right, int range\_left);

/\*

\* @brief Пользователь заполняет массив

\* @param size - размер массива

\* @param array - массив

\* @param range\_right - правая граница диапазона

\* @param range\_left - левая граница диапазона

\* @return 0 - в случае успеха

\*/

int fill\_by\_your\_self(const size\_t size, int\* array, int range\_right, int range\_left);

/\*

\* @brief чистит использованную память

\* @param array - массив

\*/

void free\_array(int\* array);

/\*

\* @brief печатает элементы массива

\* @param array - массив

\* @param size - размер массива

\*/

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

/\*

\* @brief Точка входа в программу

\* @return 0, в случае успешного завершения программы

\*/

int main() {

const size\_t size = get\_size\_t(get\_value("Введите размер массива\t"));

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

int range\_left = get\_value("\nВведите левую границу диапозона\t");

int range\_right = get\_value("\nВведите правую границу диапозона\t");

check\_range(range\_right, range\_left);

int choice\_1 = get\_choice\_fill();

enum fill\_in fill\_in = (enum fill\_in)choice\_1;

switch (fill\_in)

{

case (fill\_randomm):

fill\_random(size, array, range\_right, range\_left);

puts("\nМассив заполнен рандомными числами:\n");

break;

case (fill\_by\_my\_self):

fill\_by\_your\_self(size, array, range\_right, range\_left);

puts("\nВы заполнили массив:\n");

break;

default:

printf\_s("Вы ввели неправильное значение\n");

errno = EIO;

return 1;

}

print\_array(array, size);

int choice\_2 = get\_choice\_action();

enum action action = (enum action)choice\_2;

int\* B = NULL;

switch (action)

{

case (sum\_10):

printf\_s("Сумма отрицательных чисел, кратных 10 = %d\n", get\_sum\_10(array, size));

break;

case (exchange):

printf\_s("\nМассив до:\t");

print\_array(array, size);

B = exchange\_array(array, size);

printf\_s("\nМассив после:\t");

print\_array(B, size);

free\_array(B);

break;

case (para):

if (is\_exist\_para(array, size))

{

puts("\nСуществует такая пара элементов с произедением, равным заданному числу\n");

}

else

{

puts("\nНе существует такой пары элементов с произедением, равным заданному числу\n");

}

break;

default:

errno = EIO;

perror("Вы ввели неверное значение");

return 1;

}

free\_array(array);

puts("\nGame over");

return 0;

}

int get\_sum\_10(int\* array, size\_t size)

{

int sum = 0;

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

{

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

{

sum += array[i];

}

}

return sum;

}

int\* exchange\_array(int\* array, size\_t size)

{

int\* B = get\_array(size);

for (int m = 0; m < size; m++)

{

B[m] = array[m];

}

size\_t k = get\_size\_t(get\_value("\nСколко элементов хотите поменять?\t"));

assert(k <= size);

size\_t l = k / 2;

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

{

swap(&(B[i]), &(B[k - i - 1]));

}

return B;

}

void swap(int\* a, int\* b)

{

int temp = \*a;

\*a = \*b;

\*b = temp;

}

bool is\_exist\_para(int\* array, size\_t size)

{

int number = get\_value("Введите число\t");

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

{

if (array[i] \* array[i + 1] == number)

{

return true;

}

}

return false;

}

int get\_choice\_fill()

{

printf\_s("\n\n%d - Заполнить массив рандомными числами\n%d - заполнить массив самостоятельно\n\n", fill\_randomm,fill\_by\_my\_self);

int choice = get\_value("Выберите действие:\t");

return choice;

}

int get\_choice\_action()

{

printf\_s("\n\n%d - Найти сумму отрицательных элементов, значения которых кратно 10\n%d - Заменить первые k элементов массива на те же элементы в обратном порядке\n%d - Определить, есть ли пара соседних элементов с произведением, pавным заданном числу\n\n", sum\_10, exchange, para);

int choice = get\_value("Выберите действие:\t");

return choice;

}

int get\_value(const char\* massage)

{

int value = 0;

printf\_s("%s", massage);

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

if (res != 1)

{

errno = EIO;

perror("ERROR");

abort();

}

return value;

}

size\_t get\_size\_t(int value)

{

if (value <= 0)

{

errno = ENOMEM;

perror("wrong size");

abort();

}

return (size\_t)value;

}

void check\_range(int range\_right, int range\_left)

{

if (range\_left > range\_right)

{

errno = EIO;

perror("ERROR");

abort();

}

}

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

{

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

if (NULL == array)

{

errno = ENOMEM;

perror("Память не выделена\n");

abort();

}

return array;

}

int fill\_random(const size\_t size, int\* array, int range\_right, int range\_left)

{

unsigned int ttime = (unsigned int)time(NULL);

srand(ttime);

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

array[i] = rand() % (range\_right - range\_left + 1) + range\_left;

}

return 0;

}

int fill\_by\_your\_self(const size\_t size, int\* array, int range\_right, int range\_left)

{

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

{

int element = get\_value("Введите элемент массива в введенном диапазоне:\t");

if (element > range\_right || element < range\_left)

{

perror("ERROR");

errno = EIO;

abort();

}

array[i] = element;

}

return 0;

}

void free\_array(int\* array)

{

if (NULL != array)

{

free(array);

}

}

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

{

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

{

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

}

}

* 1. Результаты выполнения программы 4.1

Результаты выполнения программы представлены ниже (Рисунок 16, Рисунок 17, Рисунок 18).

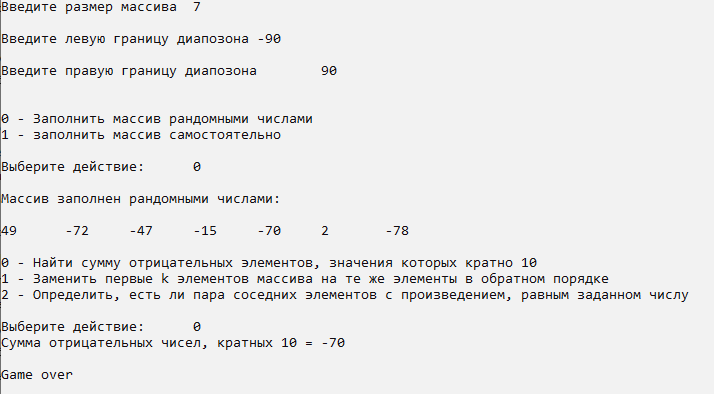


Рисунок - Результат выполнения первого пункта программы

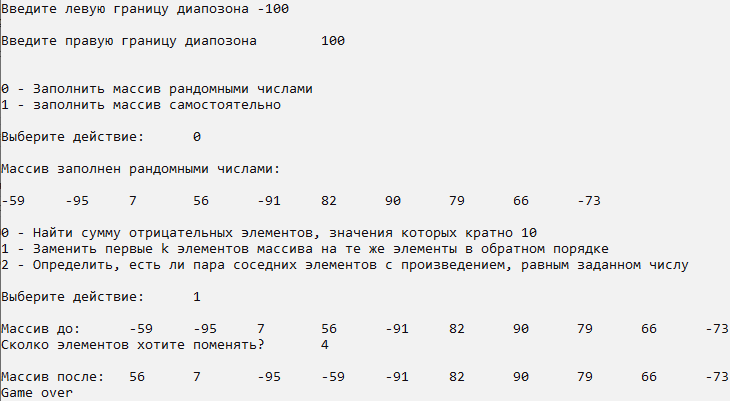


Рисунок - Результат выполнения второго пункта программы

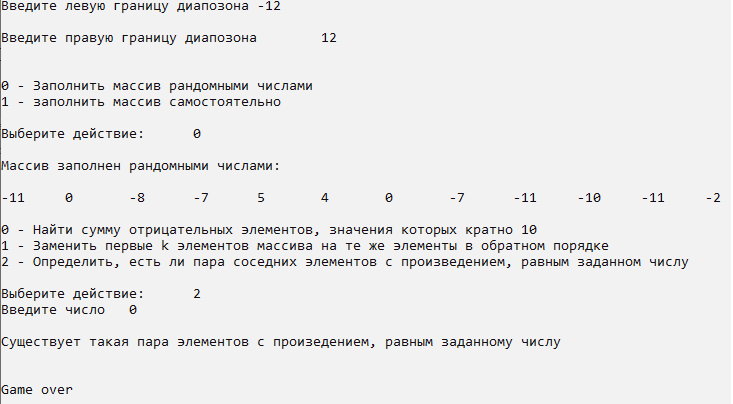


Рисунок - Результат выполнения третьего пункта программы

* 1. Отметка о выполнении задания в веб-хостинге системы контроля версий 4.1

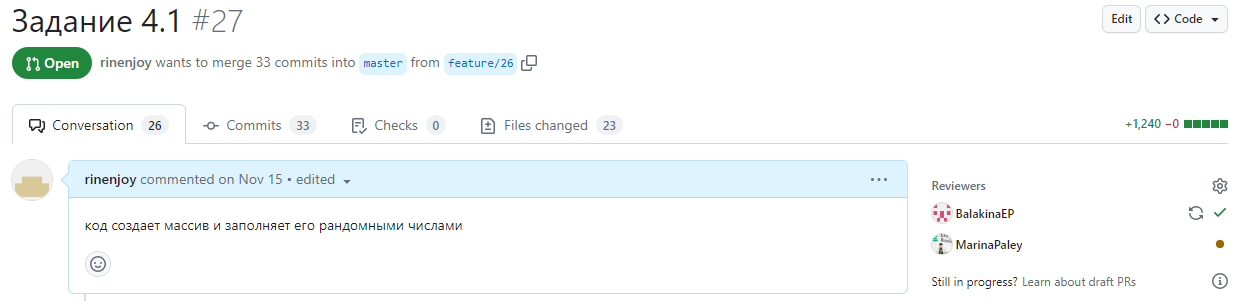


Рисунок - Approved task 4.1

1. Задача 4.2
   1. Формулировка задания 4.2

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 - Исходные данные

|  |  |  |
| --- | --- | --- |
| Вариант | Задачи | Интервал |
| 6 | 1. Заменить минимальный элемент массива на средний (количество элементов – нечетно). 2. Удалить из него все элементы, в записи которых есть цифра 5. 3. Из элементов массива C сформировать массив A той же размерности по правилу: если номер i элемента четный, то Ai=Ci2, если нечетный, то Ai=2Ci. | [-1000;1000] |

* 1. Блок-схема алгоритма 4.2

Блок-схема основного алгоритма представлена ниже (). Блок-схемы используемых функций представлены ниже ().

* 1. Текст программы на языке C 4.2

#include <stdio.h>

#include <stdlib.h>

#include <errno.h>

#include <time.h>

#include <stdbool.h>

/\*

\* @brief Выбор пользователя

\* @brief fill\_random - Заполнение массива пользователем

\* @brief fill\_by\_my\_self - Заполнение массива рандомными числами

\*/

enum fill\_in {fill\_randomm, fill\_by\_my\_self};

/\*

\* @brief Выбор пользователя

\* @brief min\_middle\_change Заменить минимальный элемент массива на средний (количество элементов – нечетно)

\* @brief delete\_element\_with\_5 Удалить из массива все элементы, в записи которых есть цифра 5

\* @brief new\_array Из элементов массива C сформировать массив A той же размерности по правилу: если номер i элемента четный, то Ai=Ci2, если нечетный, то Ai=2Ci.

\*/

enum action {min\_middle\_change, delete\_element\_with\_5, new\_array};

/\*

\* @brief сортирует массив

\* @param array - массив

\* @param size - размер массива

\*/

void bubble\_sort(int\* array, size\_t size);

/\*

\* @brief меняет два элемента местами

\* @param a - элемент массива

\* @param b - элемент массива

\*/

void swap(int\* a, int\* b);

/\*

\* @brief меняет минимальный элемент на средний

\* @param array - массив

\* @param size - размер массива

\* @return изменённый массив

\*/

int\* min\_middle(int\* array, size\_t size);

/\*

\* @brief Ищет минимальный элемент массива

\* @param array - массив

\* @param size - размер массива

\* @return минимальный элемент

\*/

int min\_element(int\* array, size\_t size);

/\*

\* @brief Ищет средний элемент массива

\* @param array - массив

\* @param size - размер массива

\* @return средний элемент

\*/

int middle\_element(int\* array, size\_t size);

/\*

\* @brief копирует массив

\* @param array - массив

\* @param size - размер массива

\* @return скопироанный массив

\*/

int\* copy\_array(int\* array, size\_t size);

/\*

\* @brief считает сколько элементов с цифрой 5 в числе

\* @param array - массив

\* @param size - размер массива

\* @return кол-во элементов с цифрой 5 в числе

\*/

int amount\_5(int\* array, size\_t size);

/\*

\* @brief Определяет, есть ли цифра пять в числе

\* @param а - элемент массива(число)

\* @return true, если цифра пять есть в числе

\*/

bool has\_five(int a);

/\*

\* @brief создает и заполняет массив элементами без цифры 5

\* @param array - массив

\* @param size - размер массива

\* @param k - кол-во элементов с цифрой 5 в числе

\* @return заполненный массив

\*/

int\* get\_array\_5(int\* array, size\_t k, size\_t size);

/\*

\* @brief предлагает выбор пользователю

\* @return введенный номер действия

\*/

int get\_choice\_fill();

/\*

\* @brief предлагает выбор пользователю

\* @return введенный номер действия

\*/

int get\_choice\_action();

/\*

\* @brief проверяет введенное значение

\* @param massage - сообщение для пользователя

\* @return переменную

\*/

int get\_value(const char\* massage);

/\*

\* @brief проверяет чтобы введенное значение было > 0 и нечетное

\* @param value введенное значение

\* @return размер массива

\*/

size\_t get\_size(int value);

/\*

\* @brief создает массив

\* @param size - размер массива

\* @return массив

\*/

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

/\*

\* @brief создает и заполняет массив по правилу: если номер i элемента четный, то Ai=Ci2, если нечетный, то Ai=2Ci

\* @param array - массив

\* @param size - размер массива

\* @return новый массив

\*/

int\* get\_new\_array(int\* array, const size\_t size);

/\*

\* @brief Заполняет массив рандомными числами

\* @param size - размер массива

\* @param array - массив

\* @return 0 - в случае успеха

\*/

int fill\_random(const size\_t size, int\* array);

/\*

\* @brief Проверяет правильность введеного диапазона

\* @param left - леая граница диапазона

\* @param right - правая граница диапазона

\*/

void check\_range(int left, int right);

/\*

\* @brief Пользователь заполняет массив

\* @param size - размер массива

\* @param array -массив

\*/

void fill\_by\_your\_self(const size\_t size, int\* array);

/\*

\* @brief чистит использованную память

\* @param array - массив

\*/

void free\_array(int\* array);

/\*

\* @brief печатает элементы массива

\* @param array - массив

\* @param size - размер массива

\*/

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

/\*

\* @brief Точка входа в программу

\* @return 0, в случае успешного завершения программы

\*/

int main() {

const size\_t size = get\_size(get\_value("Введите размер массива\t"));

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

enum fill\_in fill\_in = (enum fill\_in)get\_choice\_fill();

switch (fill\_in)

{

case (fill\_randomm):

fill\_random(size, array);

puts("\nМассив заполнен рандомными числами:\n");

break;

case (fill\_by\_my\_self):

fill\_by\_your\_self(size, array);

puts("\nВы заполнили массив:\n");

break;

default:

errno = EIO;

perror("Вы ввели неверное значение");

return 1;

}

print\_array(array, size);

enum action action = (enum action)get\_choice\_action();

int\* new = NULL;

switch (action)

{

case (min\_middle\_change):

printf\_s("\n\n%d - Минимальный элемент\n%d - Средний элемент\n", min\_element(array, size), middle\_element(array, size));

puts("\nМассив после:\t");

new = min\_middle(array, size);

print\_array(new, size);

free\_array(new);

break;

case (delete\_element\_with\_5):

puts("\nМассив до:\t");

print\_array(array, size);

puts("\nМассив после:\t");

new = get\_array\_5(array, amount\_5(array, size), size);

print\_array(new, (size - amount\_5(array, size)));

free\_array(new);

break;

case (new\_array):

new = get\_new\_array(array, size);

print\_array(new, size);

free\_array(new);

break;

default:

errno = EIO;

perror("Вы ввели неверное значение");

return 1;

}

free\_array(array);

puts("\n\nGame over");

return 0;

}

int get\_choice\_fill()

{

printf\_s("\n\n%d - Заполнить массив рандомными числами\n%d - заполнить массив самостоятельно\n", fill\_randomm, fill\_by\_my\_self);

int choice = get\_value("\nВыберите действие:\t");

return choice;

}

int get\_choice\_action()

{

printf\_s("\n%d - Заменить минимальный элемент массива на средний(для выполнения этого задания размер массива должен быть нечетным числом)\n%d - Удалить из массива все элементы, в записи которых есть цифра 5\n%d - Из элементов массива C сформировать массив A той же размерности по правилу: если номер i элемента четный, то Ai=Ci^2, если нечетный, то Ai=2Ci.\n", min\_middle\_change, delete\_element\_with\_5, new\_array);

int choice = get\_value("\nВыберите действие:\t");

return choice;

}

void bubble\_sort(int\* array, size\_t size)

{

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

{

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

{

if (array[j] > array[j + 1])

{

swap(&(array[j]), &(array[j + 1]));

}

}

}

}

void swap(int\* a, int\* b)

{

int temp = \*a;

\*a = \*b;

\*b = temp;

}

int\* min\_middle(int\* array, size\_t size)

{

if (size % 2 == 1)

{

int\* B = copy\_array(array, size);

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

{

if (B[j] == min\_element(B, size))

{

B[j] = middle\_element(B, size);

break;

}

}

return B;

}

else

{

errno = EIO;

perror("\nНеверный размер(должен быть нечетным числом)");

abort();

}

}

int min\_element(int\* array, size\_t size)

{

int min = array[0];

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

{

if (array[i] < min)

{

min = array[i];

}

}

return min;

}

int middle\_element(int\* array, size\_t size)

{

int\* B = copy\_array(array, size);

bubble\_sort(B, size);

int middle = B[(size / 2)];

free\_array(B);

return middle;

}

int\* copy\_array(int\* array, size\_t size)

{

int\* B = get\_array(size);

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

{

B[i] = array[i];

}

return B;

}

int amount\_5(int\* array, size\_t size)

{

int k = 0;

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

{

if (has\_five(array[i]))

{

k++;

}

}

return k;

}

bool has\_five(int a)

{

int b = abs(a);

while (b > 0)

{

if (b % 10 == 5)

{

return true;

}

b = b / 10;

}

return false;

}

int\* get\_array\_5(int\* array, size\_t k, size\_t size)

{

int\* array\_without\_5 = get\_array(size - k);

size\_t i = 0;

size\_t j = 0;

while (i < size)

{

if (has\_five(array[i]))

{

i++;

}

else

{

array\_without\_5[j] = array[i];

i++;

j++;

}

}

return array\_without\_5;

}

int get\_value(const char\* massage)

{

int value = 0;

printf\_s("%s", massage);

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

if (res != 1)

{

errno = EIO;

perror("ERROR");

abort();

}

return value;

}

size\_t get\_size(int value)

{

if (value <= 0)

{

errno = ENOMEM;

perror("Wrong size");

abort();

}

return (size\_t)value;

}

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

{

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

if (NULL == array)

{

errno = ENOMEM;

perror("Память не выделена\n");

abort();

}

return array;

}

int\* get\_new\_array(int\* array, const size\_t size)

{

int\* new\_array = get\_array(size);

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

{

if (i % 2 == 0)

{

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

}

else

{

new\_array[i] = 2 \* array[i];

}

}

return new\_array;

}

int fill\_random(const size\_t size, int\* array)

{

unsigned int ttime = (unsigned int)time(NULL);

srand(ttime);

int A = get\_value("\nВведите левую границу диапозона\t");

int B = get\_value("\nВведите правую границу диапозона\t");

check\_range(A, B);

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

array[i] = rand() % (B - A + 1) + A;

}

return 0;

}

void check\_range(int left, int right)

{

if (left > right)

{

errno = EIO;

perror("\nWrong number");

abort();

}

}

void fill\_by\_your\_self(const size\_t size, int\* array)

{

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

{

array[i] = get\_value("Введите элемент массива\t");

}

}

void free\_array(int\* array)

{

if (NULL != array)

{

free(array);

}

}

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

{

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

{

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

}

}

* 1. Результаты выполнения программы 4.2

Результат выполнения программы представлен ниже (Рисунок 20, Рисунок 21, Рисунок 22).

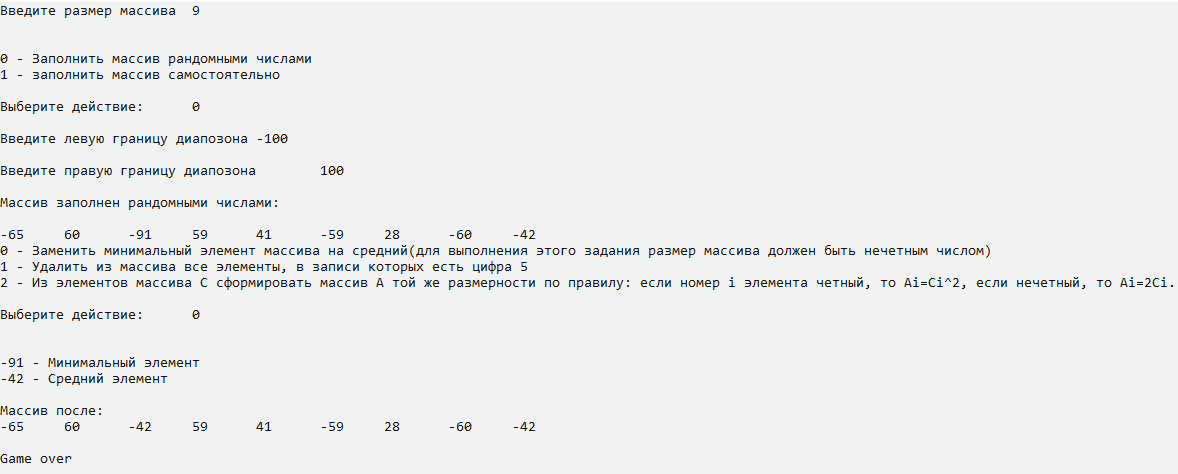


Рисунок 20 - Результат выполнения первого пункта программы

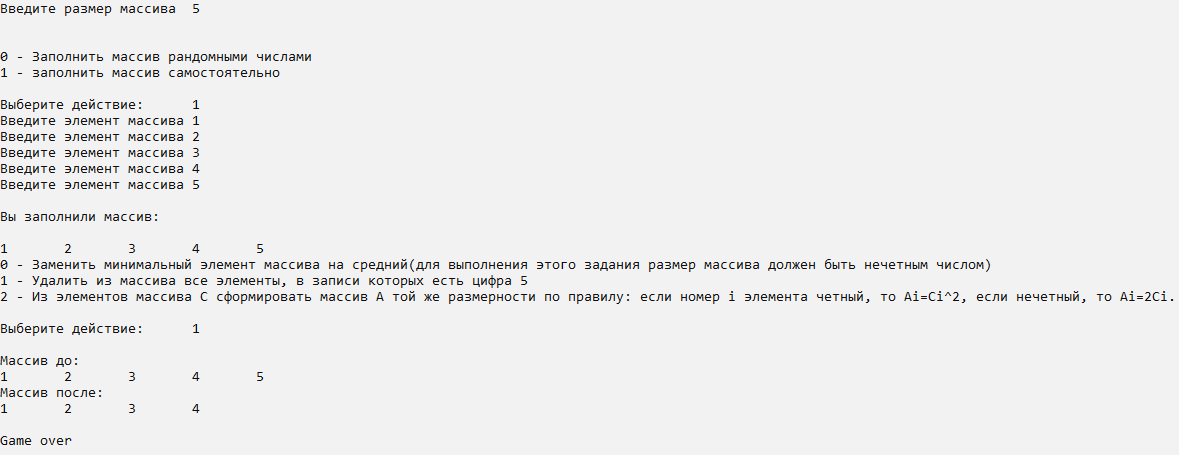


Рисунок 21 - Результат выполнения второго пункта программы

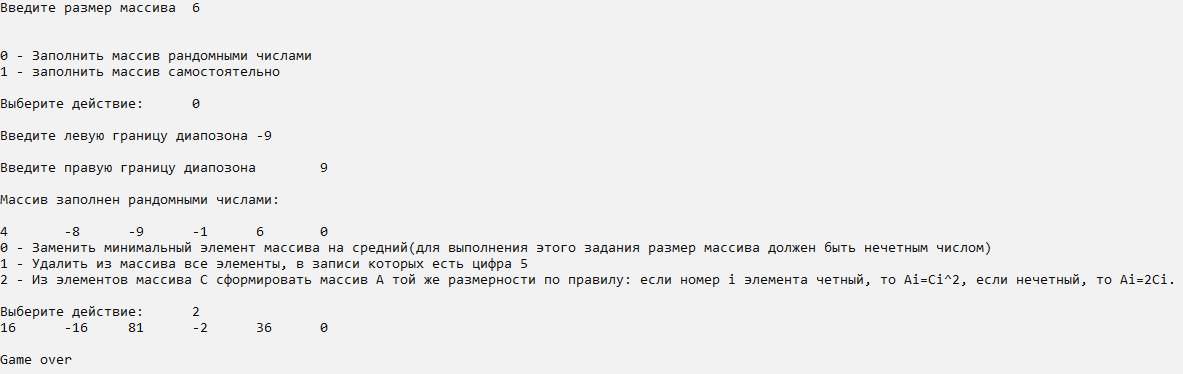


Рисунок 22 - Результат выполнения третьего пункта программы

* 1. Отметка о выполнении задания в веб-хостинге системы контроля версий 4.2

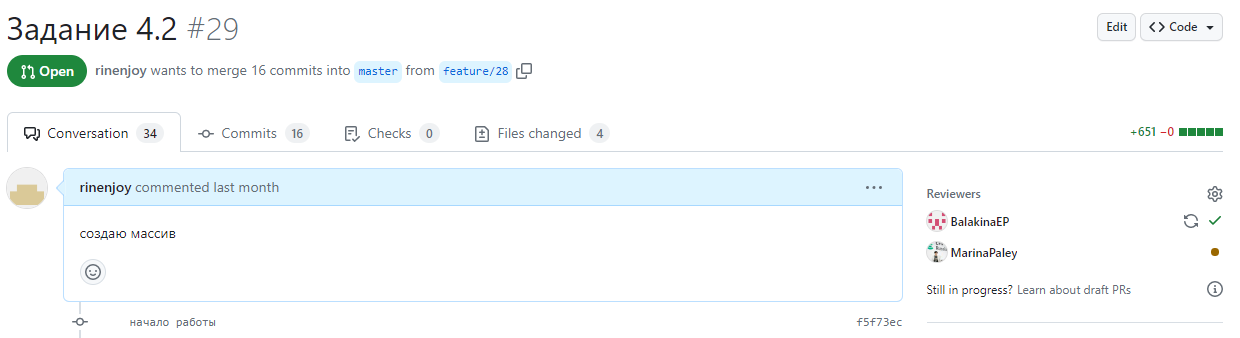


Рисунок - Approved task 4.2

1. Задача 4.3
   1. Формулировка задания 4.3

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 - Исходные данные

|  |  |
| --- | --- |
| Вариант | Задачи |
| 6 | 1. Заменить максимальный элемент каждой строки нулем. 2. Вставить перед всеми строками, первый элемент которых делится на 3, строку из нулей. |

* 1. Блок-схема алгоритма 4.3

Блок-схема основного алгоритма представлена ниже (). Блок-схемы используемых функций представлены ниже ().

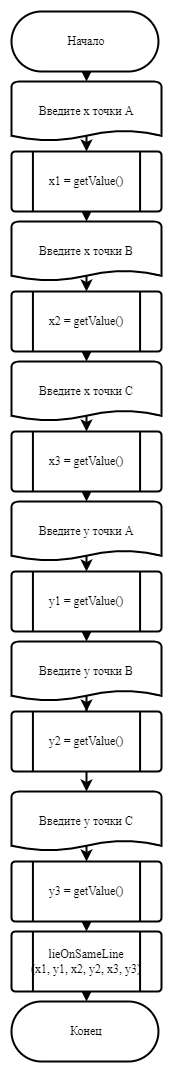


Рисунок 24 - Блок-схема основного алгоритма

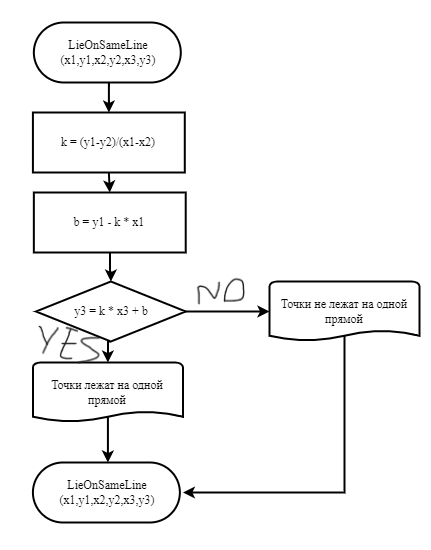


Рисунок 25 - Блок-схема используемой функции

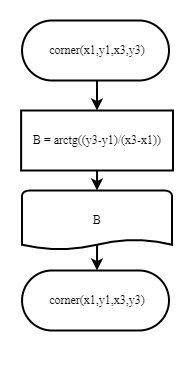


Рисунок 26 - Блок-схема используемой функции

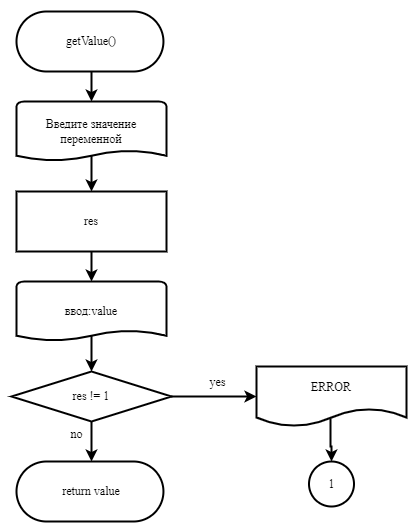


Рисунок 27 - Блок-схема используемой функции

* 1. Текст программы на языке C 4.3

#include <stdio.h>

#include <stdlib.h>

#include <errno.h>

#include <time.h>

/\*

\* @brief Выбор пользователя

\* @brief fill\_random - Заполнение массива пользователем

\* @brief fill\_by\_my\_self - Заполнение массива рандомными числами

\*/

enum fill\_in {fill\_randomm, fill\_by\_my\_self};

/\*

\* @brief Выбор пользователя

\* @brief change\_max\_zero Заменить максимальный элемент каждой строки нулем

\* @brief put\_str\_0 Вставить перед всеми строками, первый элемент которых делится на 3, строку из нулей

\*/

enum action {change\_max\_zero, put\_str\_0};

/\*

\* @brief Вставляет перед всеми строками, первый элемент которых делится на 3, строку из нулей

\* @param array - массив

\* @param line - кол-во строк

\* @param column - кол-во столбцов

\* @return изменённый массив

\*/

int\*\* new\_array\_str\_0(int\*\* array, const size\_t line, const size\_t column);

/\*

\* @brief Считает строки, первый элемент которых делится на 3

\* @param array - массив

\* @param line - кол-во строк

\* @return кол-во строк

\*/

int line\_amount\_3(const int\*\* const array, const size\_t line);

/\*

\* @brief Заменяет максимальный элемент каждой строки нулем

\* @param array - массив

\* @param line - кол-во строк

\* @param column - кол-во столбцов

\* @return изменённый массив

\*/

int\*\* max\_zero(const int\*\* const array, const size\_t line, const size\_t column);

/\*

\* @brief Ищет максимальный элемент массива

\* @param array - массив

\* @param column - размер массива

\* @return максимальный элемент

\*/

int max\_element(int\* array, size\_t column);

/\*

\* @brief Предлагает пользователю выбрать действие

\* @return введенный номер действия

\*/

int get\_choice\_fill();

/\*

\* @brief Предлагает пользователю выбрать действие

\* @return введенный номер действия

\*/

int get\_choice\_action();

/\*

\* @brief Проверяет введенное значение

\* @param massage - сообщение для пользователя

\* @return переменную

\*/

int get\_value(const char\* massage);

/\*

\* @brief Проверяет чтобы введенное значение было > 0 и нечетное

\* @param value - введенное значение

\* @return размер массива

\*/

size\_t get\_size(int value);

/\*

\* @brief Создает двумерный массив

\* @param line - кол-во строк

\* @param column - кол-во столбцов

\* @return двумерный массив

\*/

int\*\* get\_array(const size\_t line, const size\_t column);

/\*

\* @brief Создает одномерный массив(строку)

\* @param column - кол-во элементов в строке

\* @return строку

\*/

int\* get\_line\_0(const size\_t column);

/\*

\* @brief Копирует массив

\* @param array - массив

\* @param line - кол-во строк

\* @param column - кол-во столбцов

\* @return копированный массив

\*/

int\*\* copy\_array(const int\*\* const array, const size\_t line, const size\_t column);

/\*

\* @brief Заполняет массив рандомными числами

\* @param array - массив

\* @param line - кол-во строк

\* @param column - кол-во столбцов

\*/

void fill\_random(int\*\* array, const size\_t line, const size\_t column);

/\*

\* @brief Пользователь заполняет массив

\* @param array - массив

\* @param line - кол-во строк

\* @param column - кол-во столбцов

\*/

void fill\_by\_your\_self(int\*\* array, const size\_t line, const size\_t column);

/\*

\* @brief Чистит использованную память

\* @param array - массив

\* @param line - кол-во строк

\* @param column - кол-во столбцов

\*/

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

/\*

\* @brief Печатает массив

\* @param array - массив

\* @param line - кол-во строк

\* @param column - кол-во столбцов

\*/

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

/\*

\* @brief Точка входа в программу

\* @return 0, в случае успешного завершения программы

\*/

int main() {

size\_t line = get\_size(get\_value("Введите кол-во строк:\t"));

size\_t column = get\_size(get\_value("Введите кол-во столбцов:"));

int\*\* array = get\_array(line, column);

enum fill\_in fill\_in = (enum fill\_in)get\_choice\_fill();

switch (fill\_in)

{

case (fill\_randomm):

fill\_random(array, line, column);

puts("\nМассив заполнен рандомными числами:\n");

break;

case (fill\_by\_my\_self):

fill\_by\_your\_self(array, line, column);

puts("\nВы заполнили массив:\n");

break;

default:

errno = EIO;

perror("\nВы ввели неверное значение\n");

return 1;

}

print\_array(array, line, column);

enum action action = (enum action)get\_choice\_action();

int k = line\_amount\_3(array, line);

int\*\* new\_array = NULL;

switch (action)

{

case (change\_max\_zero):

puts("\nИзменённый массив:\n");

new\_array = max\_zero(array, line, column);

print\_array(new\_array, line, column);

free\_array(new\_array, line);

break;

case (put\_str\_0):

puts("\nИзменённый массив:\n");

new\_array = new\_array\_str\_0(copy\_array(array, line, column), line, column);

print\_array(new\_array, line + k, column);

free\_array(new\_array, line + k);

break;

default:

errno = EIO;

perror("\nВы ввели неверное значение");

return 1;

}

free\_array(array, line);

puts("\n\nGame over");

return 0;

}

int get\_choice\_fill()

{

printf\_s("\n\n%d - Заполнить массив рандомными числами\n%d - заполнить массив самостоятельно\n", fill\_randomm, fill\_by\_my\_self);

return get\_value("\nВыберите действие:\t");

}

int get\_choice\_action()

{

printf\_s("\n%d - Заменить максимальный элемент каждой строки нулем\n%d - Вставить перед всеми строками, первый элемент которых делится на 3, строку из нулей\n", change\_max\_zero, put\_str\_0);

return get\_value("\nВыберите действие:\t");

}

int\*\* new\_array\_str\_0(int\*\* array, const size\_t line, const size\_t column)

{

size\_t k = line\_amount\_3(array, line);

int\*\* new\_array = get\_array(line + k, column);

size\_t i = 0;

size\_t j = 0;

while (i < line && j < line + k)

{

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

{

new\_array[j] = get\_line\_0(column);

new\_array[j + 1] = array[i];

i++;

j+=2;

}

else

{

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

i++;

j++;

}

}

return new\_array;

}

int line\_amount\_3(const int\*\* const array, const size\_t line)

{

size\_t k = 0;

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

{

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

{

++k;

}

}

return k;

}

int\*\* max\_zero(const int\*\* const array, const size\_t line, const size\_t column)

{

int\*\* max\_zero\_array = copy\_array(array, line, column);

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

{

int max = max\_element(max\_zero\_array[i], column);

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

{

if (max\_zero\_array[i][j] == max)

{

max\_zero\_array[i][j] = 0;

break;

}

}

}

return max\_zero\_array;

}

int max\_element(int\* array, size\_t column)

{

int max = array[0];

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

{

if (array[i] > max)

{

max = array[i];

}

}

return max;

}

int get\_value(const char\* massage)

{

int value = 0;

printf\_s("%s", massage);

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

if (res != 1)

{

errno = EIO;

perror("ERROR");

abort();

}

return value;

}

size\_t get\_size(int value)

{

if (value <= 0)

{

errno = ENOMEM;

perror("wrong size");

abort();

}

return (size\_t)value;

}

int\*\* get\_array(const size\_t line, const size\_t column)

{

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

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

{

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

}

return array;

}

int\* get\_line\_0(const size\_t column)

{

int\* line = (int\*)malloc(column \* sizeof(int));

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

{

line[i] = 0;

}

return line;

}

int\*\* copy\_array(const int\*\* const array, const size\_t line, const size\_t column)

{

int\*\* cop\_array = get\_array(line, column);

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

{

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

{

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

}

}

return cop\_array;

}

void fill\_random(int\*\* array, const size\_t line, const size\_t column)

{

unsigned int ttime = (unsigned int)time(NULL);

srand(ttime);

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

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

{

array[i][j] = rand() % 201 - 100;

}

}

}

void fill\_by\_your\_self(int\*\* array, const size\_t line, const size\_t column)

{

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

{

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

{

array[i][j] = get\_value("Введите элемент массива\t");

}

}

}

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

{

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

{

if (NULL != array[i])

{

free(array[i]);

}

}

if (NULL != array)

{

free(array);

}

}

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

{

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

{

puts("\n");

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

{

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

}

}

puts("\n");

}

* 1. Результаты выполнения программы 4.3

Результаты выполнения программы представлены ниже (Рисунок 28, Рисунок 29).

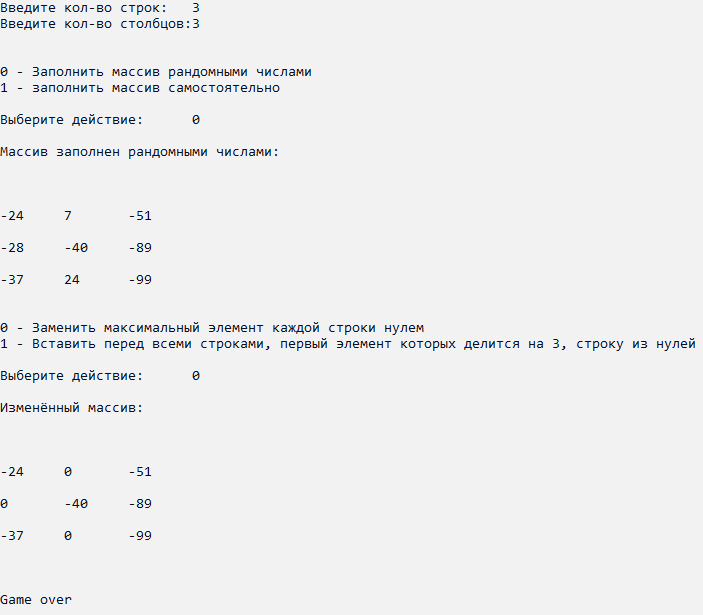


Рисунок - Результат выполнения первого пункта программы

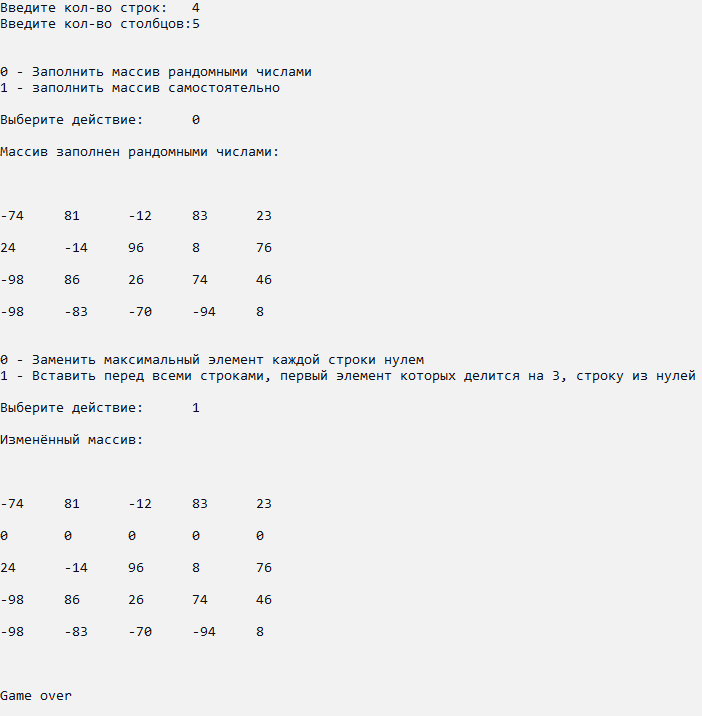


Рисунок - Результат выполнения второго пункта программы

* 1. Отметка о выполнении задания в веб-хостинге системы контроля версий 4.3

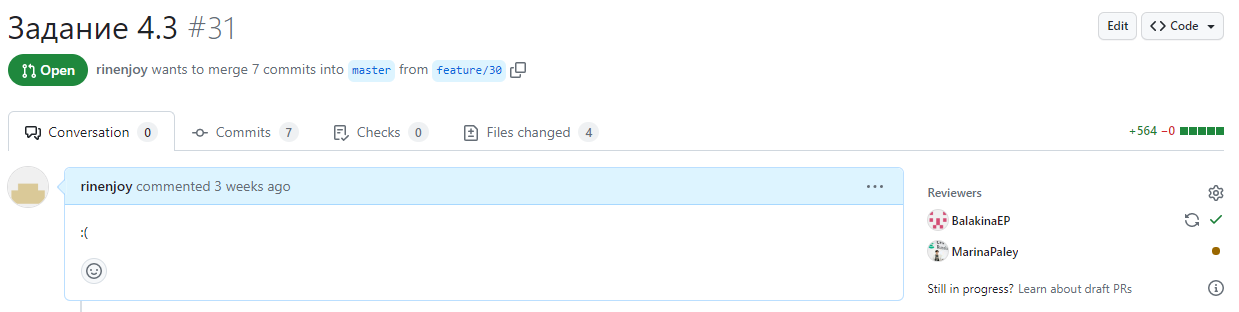


Рисунок - Approved task 4.3