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 7

Выполнил: ст. гр. ТКИ - 142

Костин Роман Сергеевич

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

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

Москва 2024

Оглавление

[1 Решение задачи 4.1 2](#_Toc156428591)

[1.1 Формулировка задачи 4.1 2](#_Toc156428592)

[1.2 Блок-схема алгоритма 3](#_Toc156428593)

[1.3 Код задания 4.1 11](#_Toc156428594)

[1.4 Решение различных тестовых примеров на C 18](#_Toc156428595)

[1.5 Подтверждение Approve 4.1 19](#_Toc156428596)

[2 решение задачи 4.2 20](#_Toc156428597)

[2.1 Формулировка задачи 4.2 20](#_Toc156428598)

[2.2 Блок-схема алгоритма 21](#_Toc156428599)

[2.3 Код задания 4.2 29](#_Toc156428600)

[2.4 Решение различных тестовых примеров на C 37](#_Toc156428601)

[2.5 Подтверждение Approve 4.2 38](#_Toc156428602)

1. Решение задачи 4.1
   1. Формулировка задачи 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  – Условия задачи 4.1

|  |  |  |
| --- | --- | --- |
| Вариант | Задача | Интервал |
| 7 | 1. Найти сумму четных отрицательных элементов. 2. Вывести индексы тех элементов, значения которых кратны 3 и 6. 3. Заменить нулями элементы массива между минимальным и максимальным, кроме их самих. | На выбор пользователя |

* 1. Блок-схема алгоритма

Блок-схема алгоритма представлена ниже (Рисунок 1, Рисунок 2, Рисунок 3, 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 16, Рисунок 17)

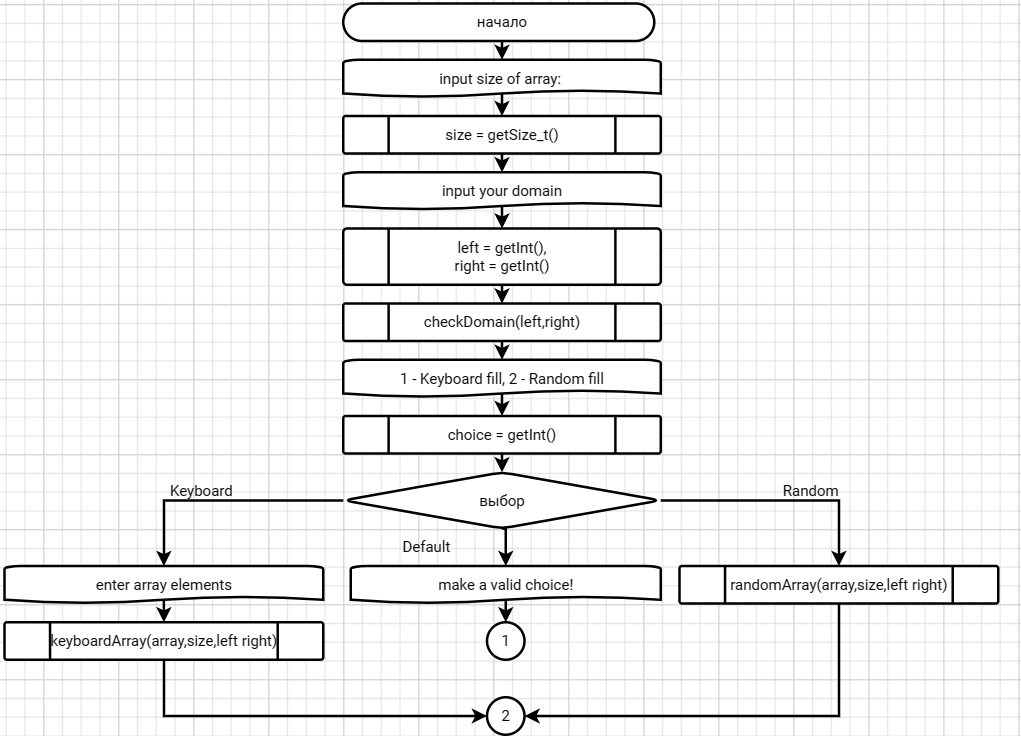


Рисунок  – Блок-схема функции main(), заполнение основного массива

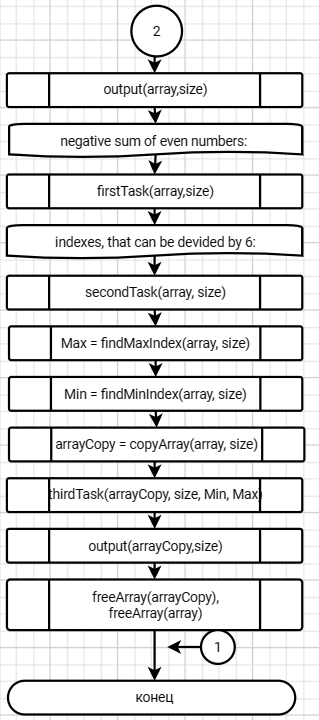


Рисунок  – Блок-схема функции main(), продолжение

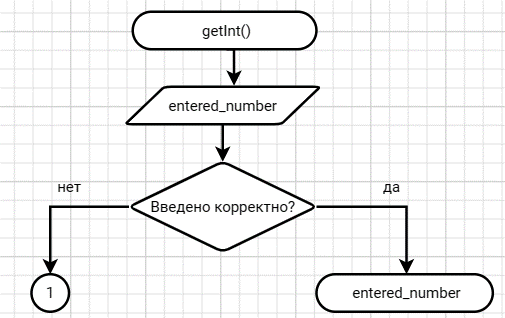


Рисунок  – Блок-схема функции getInt()

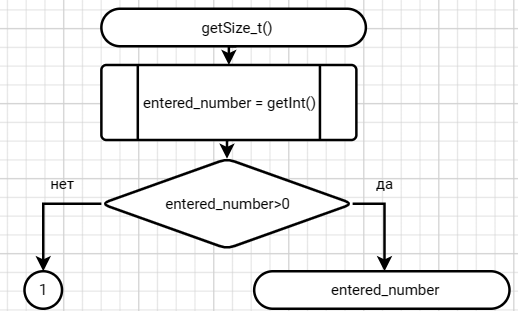


Рисунок  – Блок-схема функции getSize\_t()

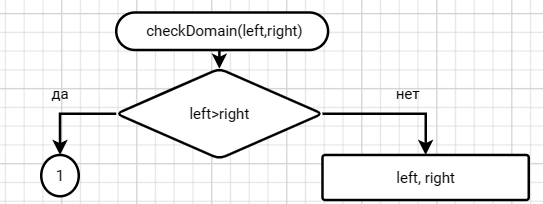


Рисунок  – Блок-схема функции checkDomain()

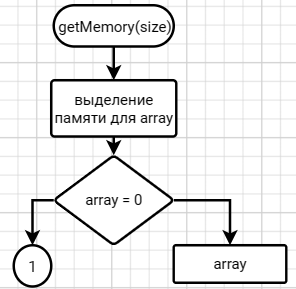


Рисунок  – Блок-схема функции getMemory()

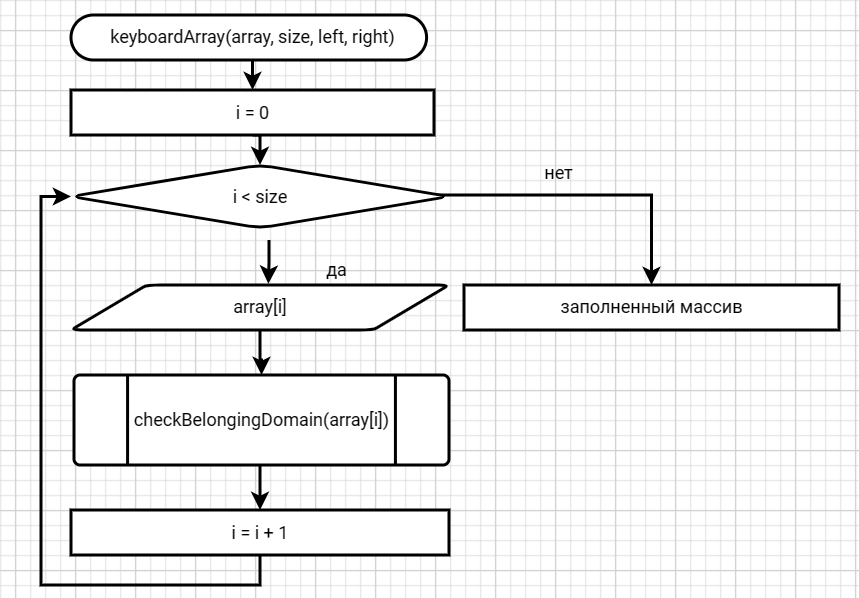


Рисунок  – Блок-схема функции keyboardArray()

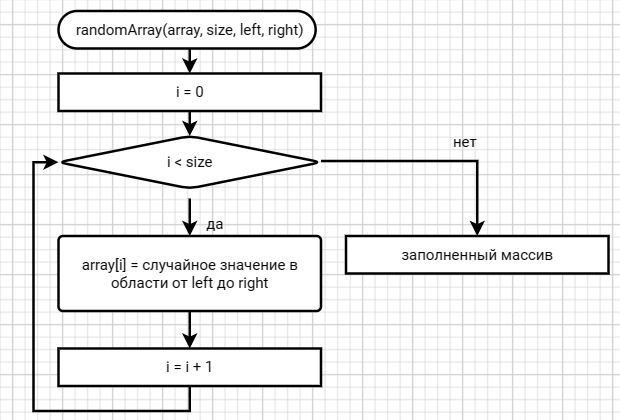


Рисунок  – Блок-схема функции randomArray()

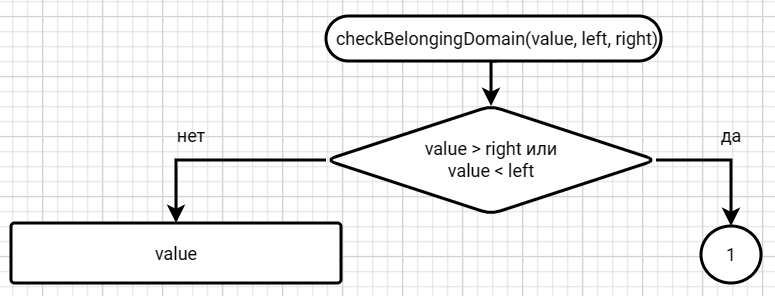


Рисунок  – Блок-схема функции checkBelongingDomain()

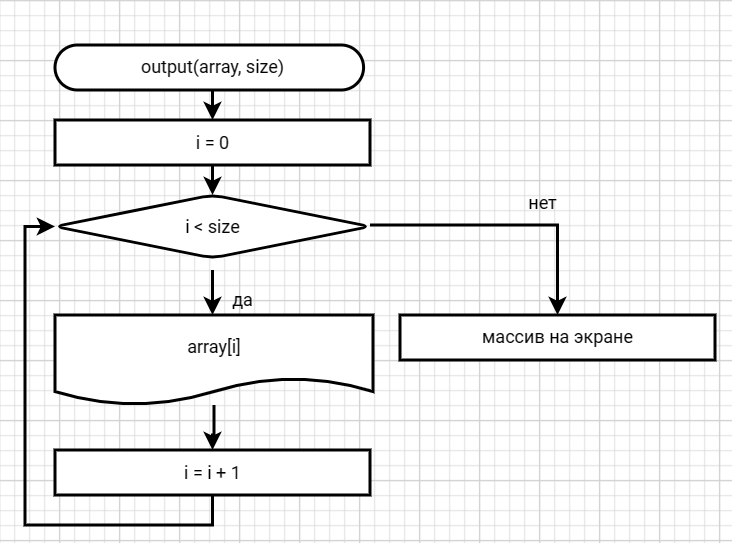


Рисунок  – Блок-схема функции output()

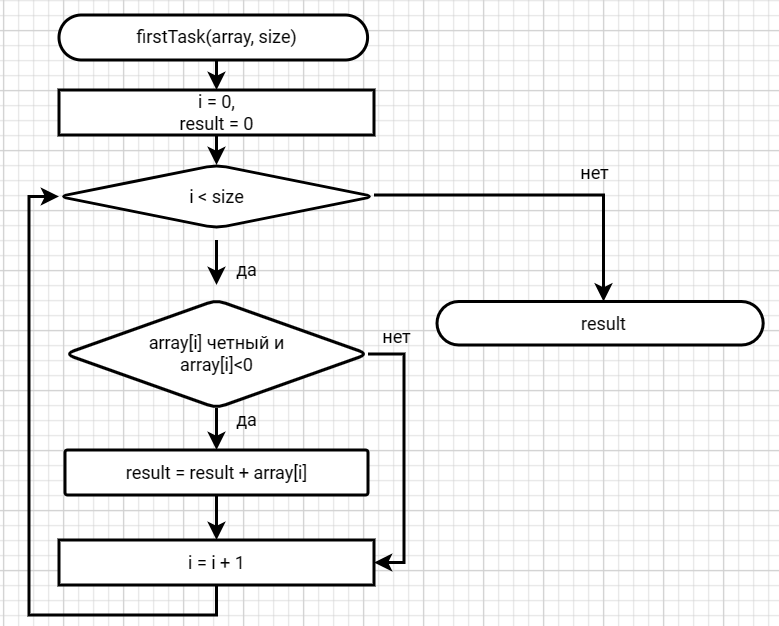


Рисунок  – Блок-схема функции firstTask()

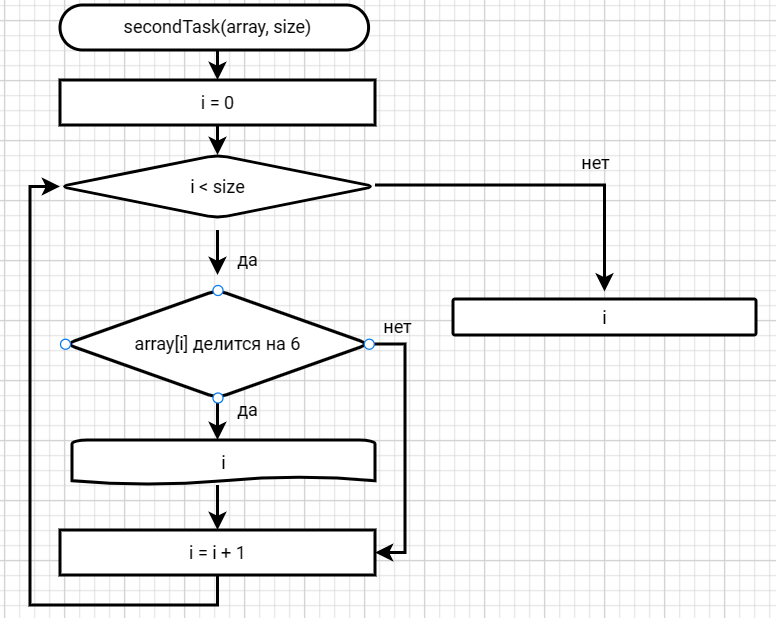


Рисунок  – Блок-схема функции secondTask()

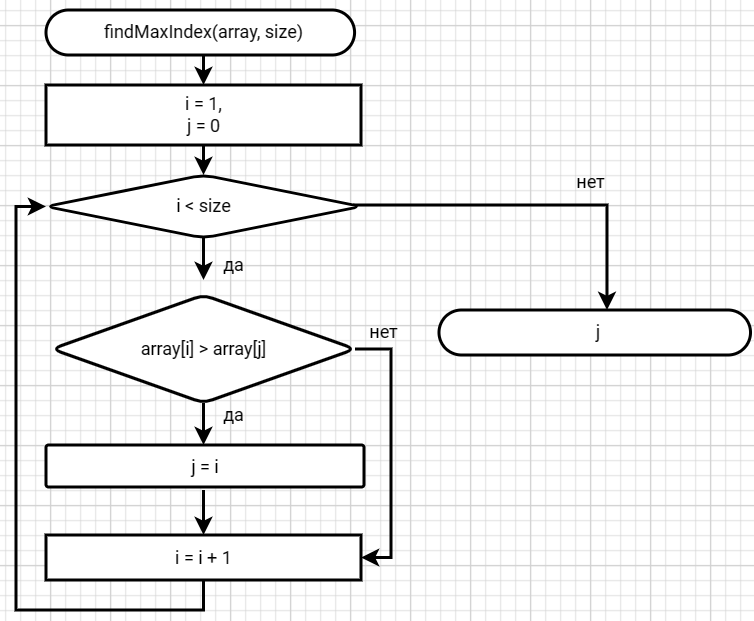


Рисунок  – Блок-схема функции findMaxIndex()

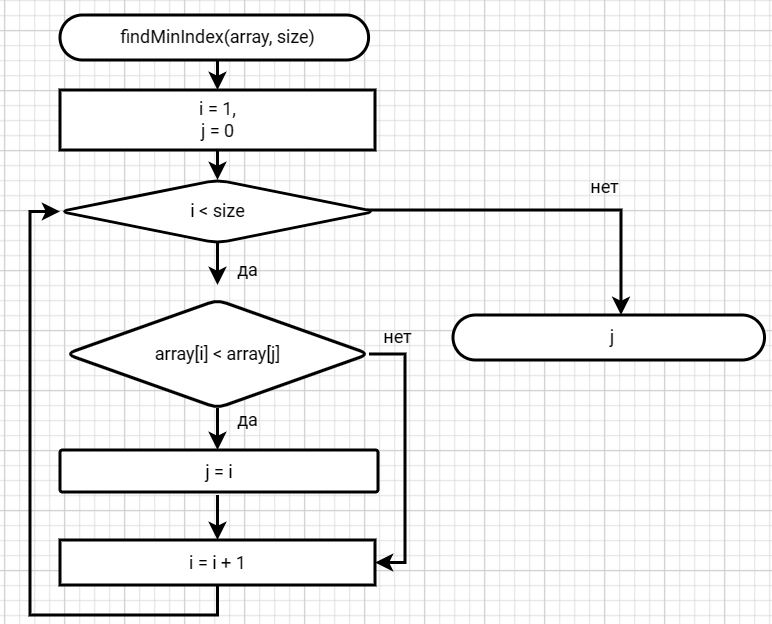


Рисунок  – Блок-схема функции findMinIndex()

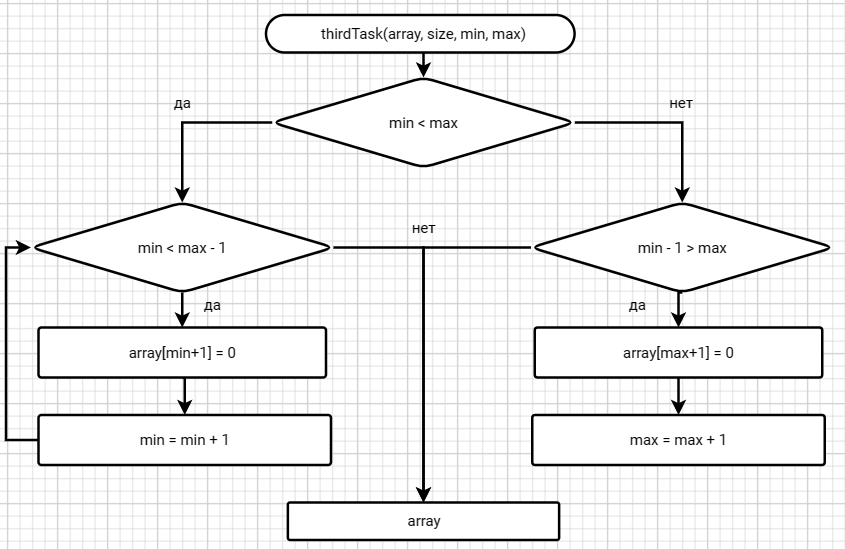


Рисунок  – Блок-схема функции thirdTask()

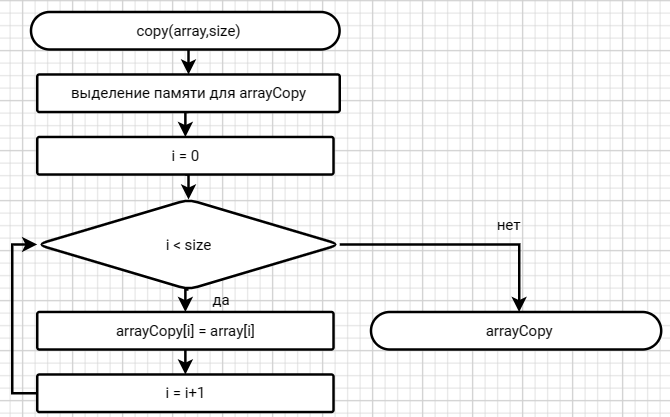


Рисунок  – Блок-схема функции copy()



Рисунок   - Блок-схема функции freeArray()

* 1. Код задания 4.1

#include <stdio.h>

#include <errno.h>

#include <stdlib.h>

#include <time.h>

/\*\*

\*@brief function find first min index of the array

\*@param array - pointer to the array

\*@param size - size of the array

\*@return j - min index

\*/

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

/\*\*

\*@brief function find first max index of the array

\*@param array - pointer to the array

\*@param size - size of the array

\*@return j - max index

\*/

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

/\*\*

\*@brief funtion checks if left border is less than right(if it false --> abort();)

\*@param left - left border of domain

\*@param right - right border of domain

\*/

void checkDomain(int const left, int const right);

/\*\*

\*@brief funtion checks if some value belongs to the domain that user chose.

\*@param value - some value

\*@param left - left border of domain

\*@param right - right border of domain

\*/

void checkBelongingDomain(int value, int const left, int const right);

/\*\*

\*@brief function for random array fill

\*@param array - pointer to array

\*@param size - size of array

\*@return arrayCopy - copy of an original array

\*/

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

/\*\*

\*@brief function for int value

\*@return entered\_number - value of entered number

\*/

int getInt();

/\*\*

\*@brief function for size\_t value

\*@return entered\_number - value of entered number

\*/

size\_t getSize\_t();

/\*\*

\*@brief function gives memory for array

\*@param size - size of array

\*@return array - pointer to an empty array

\*/

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

/\*\*

\*@brief function for filling array with user's input

\*@param array - pointer to array

\*@param size - size of array

\*@param left - left border of domain

\*@param right - right border of domain

\*/

void keyboadArray(int\* const array, const size\_t size, int const left, int const right);

/\*\*

\*@brief function for random array fill

\*@param array - pointer to array

\*@param size - size of array

\*@param left - left border of domain

\*@param right - right border of domain

\*/

void randomArray(int\* const array, const size\_t size, int const left, int const right);

/\*\*

\*@brief prints values of array

\*@param array - pointer to array

\*@param size - size of array

\*/

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

/\*\*

\*@brief function clears memory used for array

\*@param array - pointer to array

\*/

void freeArray(int\*\* array);

/\*\*

\*@brief function for the sum of all negative even numbers

\*@param array - pointer to array

\*@param size - size of array

\*@return result - sum of all negative even numbers

\*/

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

/\*\*

\*@brief function prints all values of the indexes that can be devided by 6

\*@param array - pointer to array

\*@param size - size of array

\*/

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

/\*\*

\*@brief function swaps all values of array to 0 between smallest and biggest numbers

\*@param array - pointer to array

\*@param size - size of array

\*@param minIndex - index of the first min value of an array

\*@param maxIndex - index of the first max value of an array

\*/

void thirdTask(int\* const array, size\_t const size, size\_t minIndex, size\_t maxIndex);

/\*\*

\*@brief used for choice of a way to fill an array

\*@param KEYBOARD - keyboard fill

\*@param RANDOM - random fill

\*/

enum Choices

{

KEYBOARD = 1,

RANDOM = 2,

};

/\*\*

\*@brief entry point to the programm

\*@return 0 if code works correctly, otherwise 1

\*/

int main()

{

printf\_s("Input size of array: ");

size\_t size = getSize\_t();

printf\_s("input your domain\n");

int const left = getInt(),

right = getInt();

checkDomain(left, right);

printf\_s("1 - keyboard fill; 2 - random fill ");

int choice = getInt();

int\* array = getMemory(size);

switch ((enum Choices)choice)

{

case RANDOM:

randomArray(array, size, left, right);

break;

case KEYBOARD:

printf\_s("Enter array elements:\n");

keyboadArray(array, size, left, right);

break;

default:

printf("Make a valid choice!\n");

return 1;

break;

}

output(array, size);

// Task 1

printf\_s("\n\nnegative sum of even numbers: %d\n", firstTask(array, size));

// Task 2

printf\_s("\nindexes, that can be devided by 6: \n");

secondTask(array, size);

// Task 3

printf\_s("\n\nmin index %zu; max index %zu \n", findMinIndex(array,size), findMaxIndex(array,size));

int\* arrayCopy = copy(array,size);

thirdTask(arrayCopy,size,findMinIndex(array,size),findMaxIndex(array,size));

output(arrayCopy,size);

freeArray(&arrayCopy);

freeArray(&array);

return 0;

}

void checkDomain(int const left, int const right)

{

if (left > right)

{

printf\_s("wrong domain!!!!");

abort();

}

}

void checkBelongingDomain(int value, int const left, int const right)

{

if (value<left || value>right)

{

printf\_s("value doesn't belong to domain!");

abort();

}

}

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

{

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

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

{

arrayCopy[i] = array[i];

}

return arrayCopy;

}

int getInt()

{

int entered\_number;

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

{

errno = EIO;

perror("Input Error");

abort();

}

return entered\_number;

}

size\_t getSize\_t()

{

int entered\_number = getInt();

if (entered\_number <= 0)

{

errno = ERANGE;

perror("Error: \n");

abort();

}

return (size\_t)entered\_number;

}

int\* getMemory(size\_t const size)

{

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

if (NULL == array)

{

abort();

}

return array;

}

void keyboadArray(int\* const array, const size\_t size, int const left, int const right)

{

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

{

array[i] = getInt();

checkBelongingDomain(array[i], left, right);

}

}

void randomArray(int\* const array, const size\_t size, int const left, int const right)

{

srand(time(NULL));

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

{

array[i] = rand() % (right - left + 1) + left;

}

}

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

{

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

{

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

}

}

void freeArray(int\*\* array)

{

free(\*array);

\*array = NULL;

}

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

{

int result = 0;

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

{

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

{

result += array[i];

}

}

return result;

}

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

{

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

{

if (array[i] % 6 == 0)

{

printf\_s("%zu ", i);

}

}

}

void thirdTask(int\* const array, size\_t const size, size\_t minIndex, size\_t maxIndex)

{

if (minIndex > maxIndex)

{

for (maxIndex; maxIndex < minIndex-1; maxIndex++)

{

array[maxIndex+1] = 0;

}

}

else

{

for (minIndex; minIndex < maxIndex-1; minIndex++)

{

array[minIndex+1] = 0;

}

}

}

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

{

size\_t j = 0;

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

{

if (array[j] > array[i])

{

j = i;

}

}

return j;

}

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

{

size\_t j = 0;

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

{

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

{

j = i;

}

}

return j;

}

* 1. Решение различных тестовых примеров на C

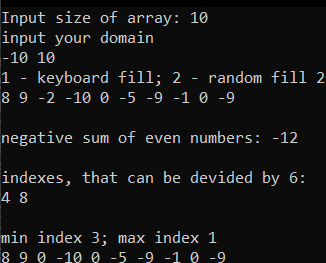


Рисунок  – Вывод программы 4.1на C при корректном вводе с случайным заполнением

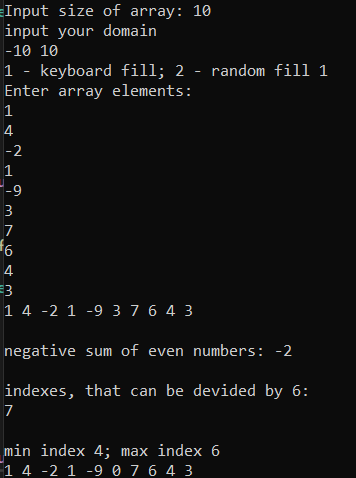


Рисунок  – Вывод программы 4.1 на C при корректном вводе с ручным заполнением



Рисунок  – Вывод программы на C при некорректном вводе размера

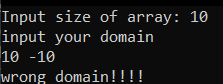


Рисунок  – Вывод программы на С при некорректном вводе области значений

* 1. Подтверждение Approve 4.1

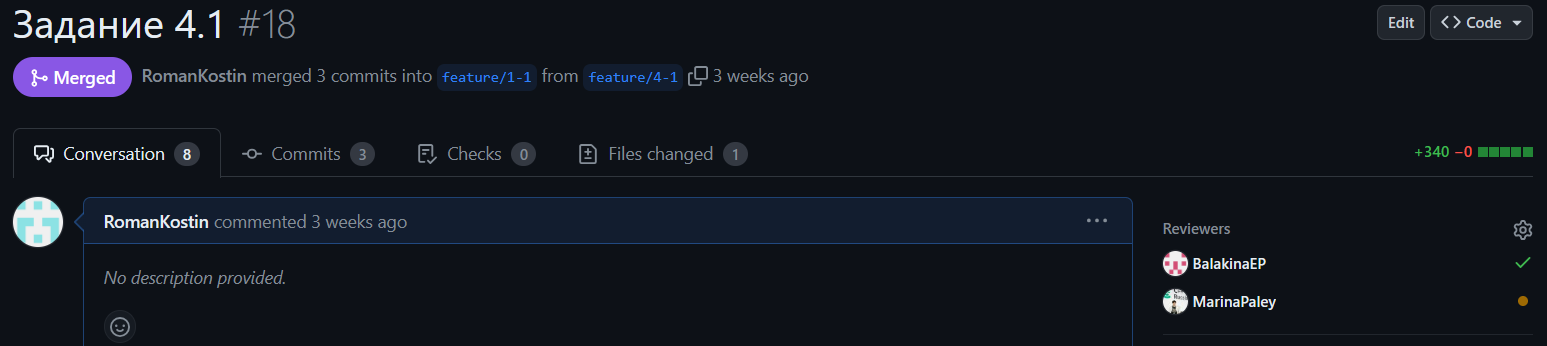


Рисунок  – Approve 4.1

1. решение задачи 4.2
   1. Формулировка задачи 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  – Условие задачи 4.2

|  |  |  |
| --- | --- | --- |
| Вариант | Задача | Интервал |
| 7 | 1. Заменить максимальный элемент массива на противоположный по знаку. 2. Вставить максимальный элемент массива после всех элементов, в которых есть цифра 1. 3. Из элементов массива C сформировать массив A той же размерности по правилу: первые N элементов находятся по формуле: Ai=Ci+i, если нечетный, то Ai=Ci-i. | На выбор пользователя |

* 1. Блок-схема алгоритма

Блок-схема алгоритма представлена ниже (Рисунок 23, Рисунок 24, Рисунок 25, Рисунок 26, Рисунок 27, Рисунок 28, Рисунок 29, Рисунок 30, Рисунок 31, Рисунок 32, Рисунок 33, Рисунок 34, Рисунок 35, Рисунок 36, Рисунок 37, Рисунок 38, Рисунок 39, Рисунок 40)

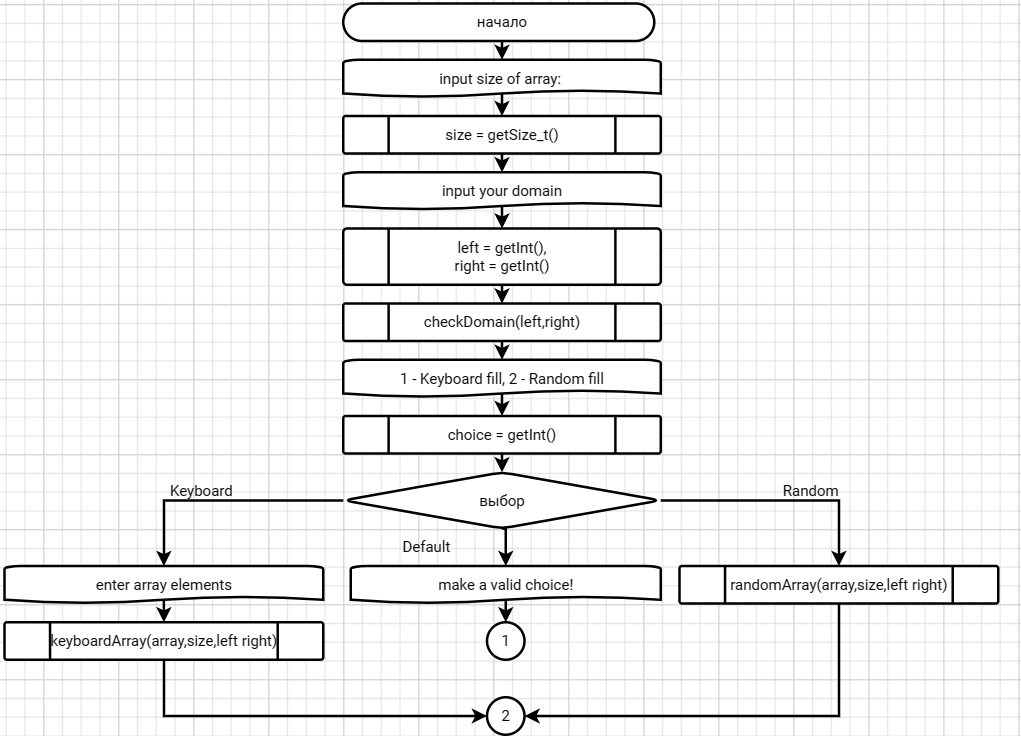


Рисунок  – Блок-схема функции main(), заполнение массива

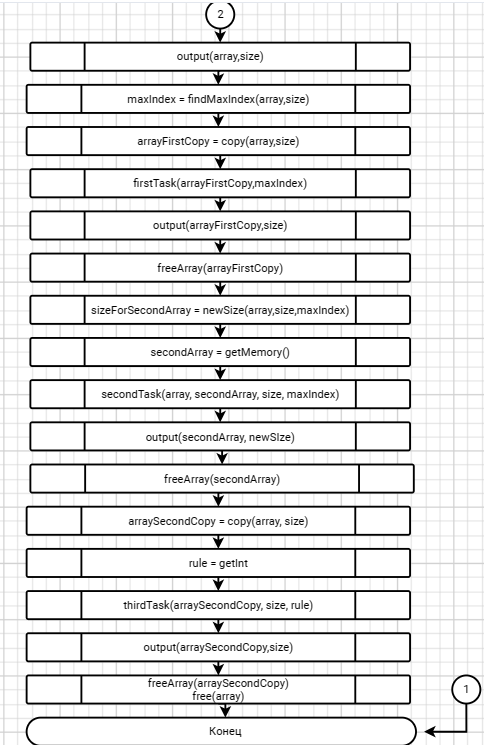


Рисунок  – Блок-схема функции main(), продолжение



Рисунок  – Блок-схема функции freeArray()

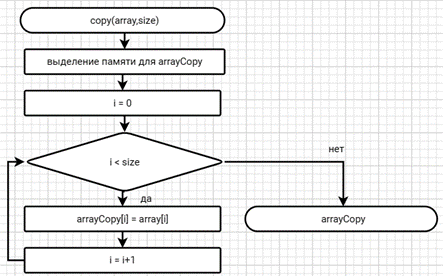


Рисунок  – Блок-схема функции copy()

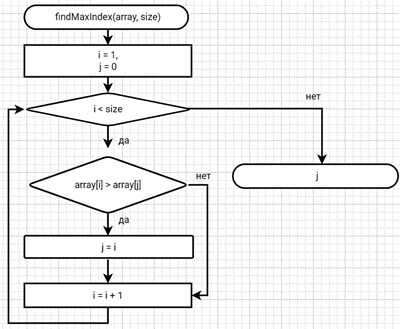


Рисунок  – Блок-схема функции findMaxIndex()

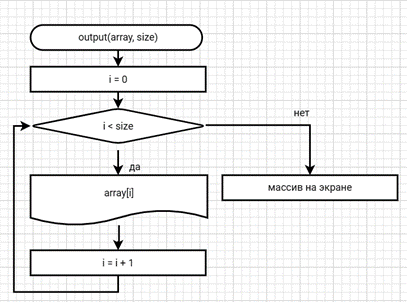


Рисунок  – Блок-схема функции output()

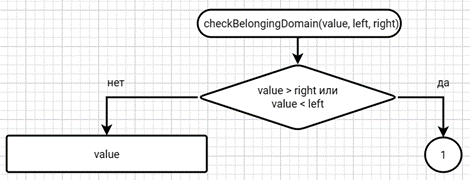


Рисунок  – Блок-схема функции checkBelongingDomain()

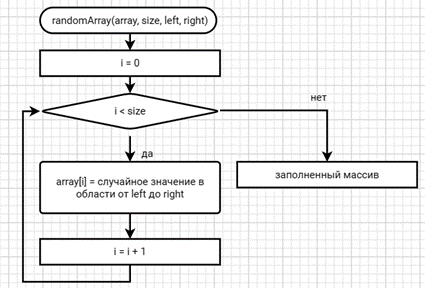


Рисунок  – Блок-схема функции randomArray()

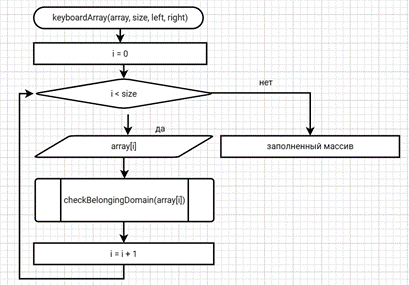


Рисунок  – Блок-схема функции keyboardArray()

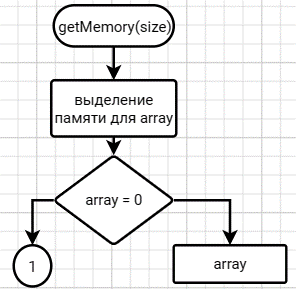


Рисунок  – Блок-схема функции getMemory()

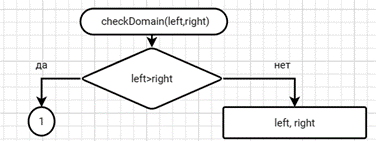


Рисунок  – Блок-схема функции checkDomain()

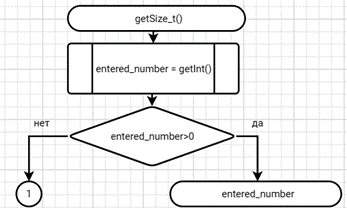


Рисунок  – Блок-схема функции getSize\_t()

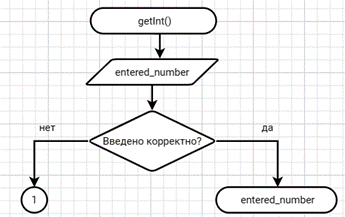


Рисунок  – Блок-схема функции getInt()

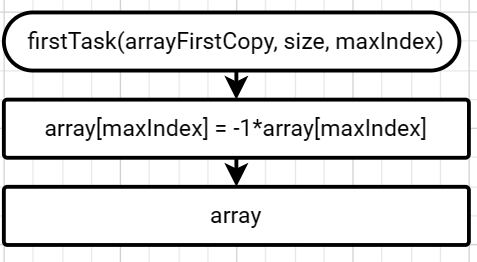


Рисунок  – Блок-схема функции firstTask()

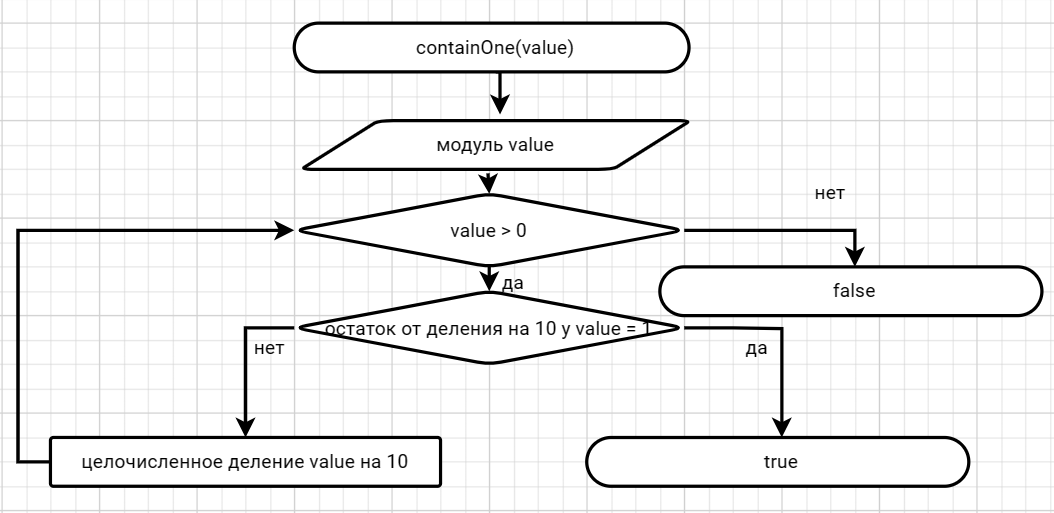


Рисунок  – Блок-схема функции containOne()

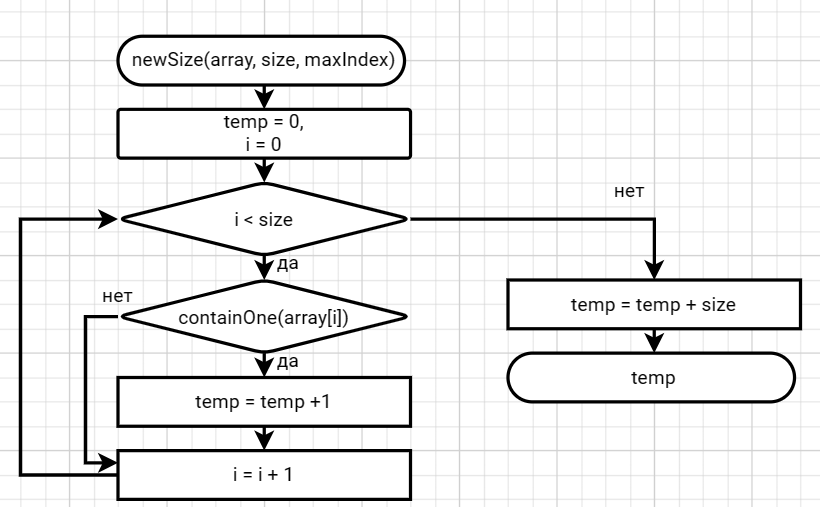


Рисунок  – Блок-схема функции newSize()

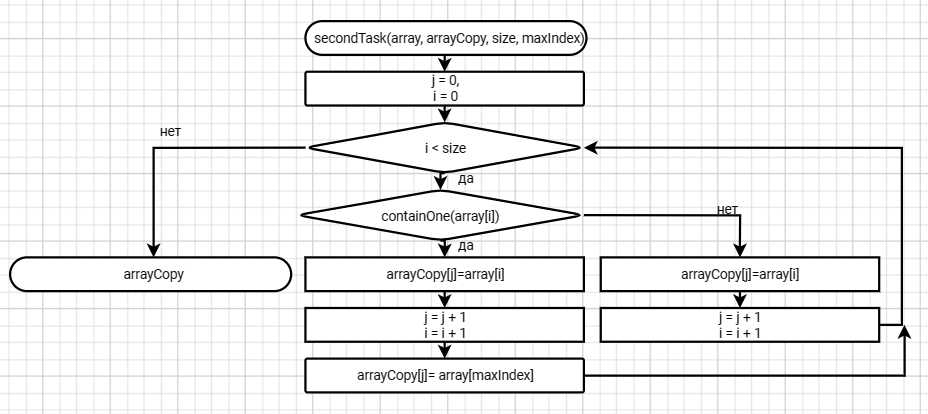


Рисунок  – Блок-схема функции secondTask()

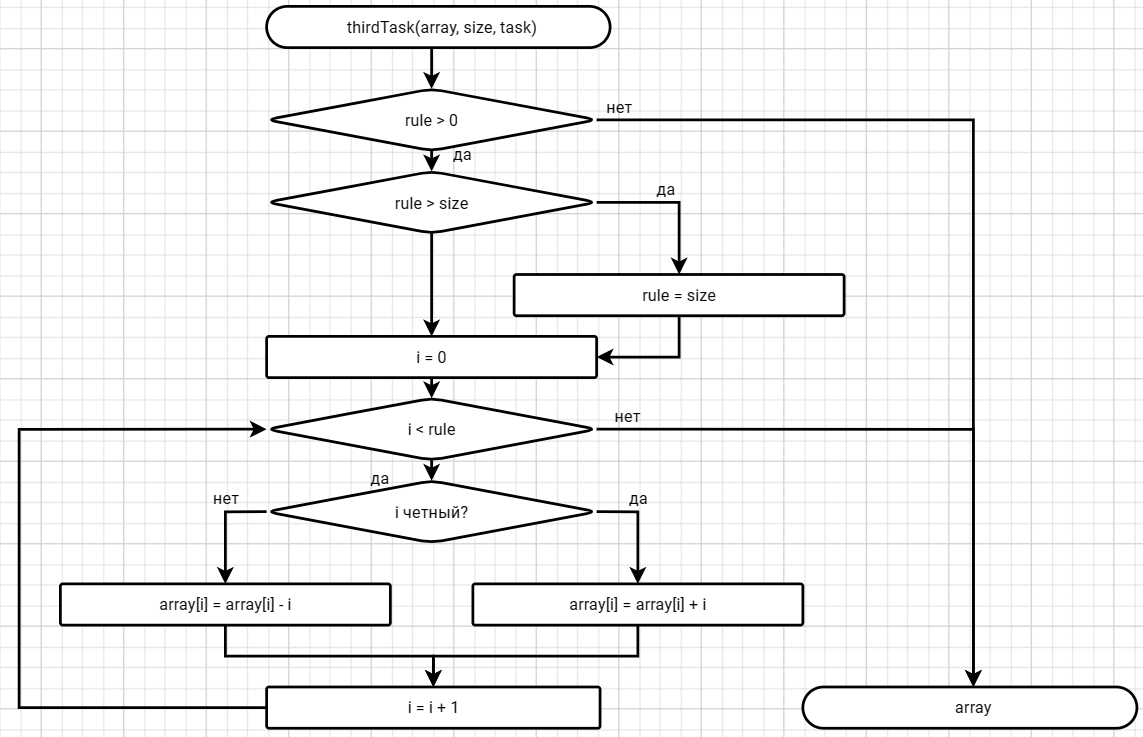


Рисунок  – Блок-схема функции thirdTask()

* 1. Код задания 4.2

#include <stdio.h>

#include <errno.h>

#include <stdlib.h>

#include <time.h>

#include <stdbool.h>

/\*\*

\*@brief function changes max value to it negative

\*@param array - pointer to the array

\*@param maxIndex - index of the max value in array

\*/

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

/\*\*

\*@brief function finds size of new array. if value from the array contain one, new size gets + 1. that rule applies to full array.

\*@param array - pointer to the array

\*@param size - size of original array

\*@param maxIndex - index of the max value in array

\*@return new size of the array

\*/

size\_t newSize(int\* const array, size\_t const size, const size\_t maxIndex);

/\*\*

\*@brief function checks if value contain one

\*@param value - some value

\*@return true - value has one; false - value do not have one

\*/

bool containOne(int value);

/\*\*

\*@brief function fills the copy of array and puts max value after each element of array that has one in it.

\*@param array - pointer to array

\*@param arrayCopy - empty array with new size

\*@param size - size of original array

\*@param maxIndex - index of the max value from original array

\*/

void secondTask(int\* const array, int\* arrayCopy, size\_t const size, size\_t maxIndex);

/\*\*

\*@brief function changes values of array with the rule that the first N elements of array changes: 1) if even A = A + index; 2) if odd A = A - index

\*@param array - pointer to the array

\*@param size - size of the array

\*@param rule - ammount of times that the rule applies

\*/

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

/\*\*

\*@brief function find first max index of the array

\*@param array - pointer to the array

\*@param size - size of the array

\*@return j - max index

\*/

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

/\*\*

\*@brief funtion checks if left border is less than right(if it false --> abort();)

\*@param left - left border of domain

\*@param right - right border of domain

\*/

void checkDomain(int const left, int const right);

/\*\*

\*@brief funtion checks if some value belongs to the domain that user chose.

\*@param value - some value

\*@param left - left border of domain

\*@param right - right border of domain

\*/

void checkBelongingDomain(int value, int const left, int const right);

/\*\*

\*@brief function for random array fill

\*@param array - pointer to array

\*@param size - size of array

\*@return arrayCopy - copy of an original array

\*/

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

/\*\*

\*@brief function for int value

\*@return entered\_number - value of entered number

\*/

int getInt();

/\*\*

\*@brief function for size\_t value

\*@return entered\_number - value of entered number

\*/

size\_t getSize\_t();

/\*\*

\*@brief function gives memory for array

\*@param size - size of array

\*@return array - pointer to an empty array

\*/

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

/\*\*

\*@brief function for filling array with user's input

\*@param array - pointer to array

\*@param size - size of array

\*@param left - left border of domain

\*@param right - right border of domain

\*/

void keyboadArray(int\* const array, const size\_t size, int const left, int const right);

/\*\*

\*@brief function for random array fill

\*@param array - pointer to array

\*@param size - size of array

\*@param left - left border of domain

\*@param right - right border of domain

\*/

void randomArray(int\* array, const size\_t size, int const left, int const right);

/\*\*

\*@brief prints values of array

\*@param array - pointer to array

\*@param size - size of array

\*/

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

/\*\*

\*@brief function clears memory used for array

\*@param array - pointer to array

\*/

void freeArray(int\*\* array);

/\*\*

\*@brief used for choice of a way to fill an array

\*@brief KEYBOARD - keyboard fill

\*@brief RANDOM - random fill

\*/

enum Choices

{

KEYBOARD = 1,

RANDOM = 2,

};

/\*\*

\*@brief entry point to the programm

\*@return 0 if code works correctly, otherwise 1

\*/

int main()

{

printf\_s("Input size of array: ");

size\_t size = getSize\_t();

printf\_s("input your domain\n");

int const left = getInt(),

right = getInt();

checkDomain(left, right);

printf\_s("%d - keyboard fill; %d - random fill ",(int)KEYBOARD, (int)RANDOM);

int choice = getInt();

int\* array = getMemory(size);

switch ((enum Choices)choice)

{

case RANDOM:

randomArray(array, size, left, right);

break;

case KEYBOARD:

printf\_s("Enter array elements:\n");

keyboadArray(array, size, left, right);

break;

default:

printf("Make a valid choice!\n");

return 1;

}

output(array, size);

size\_t maxIndex = findMaxIndex(array, size);

//Task 1

int\* arrayFirstCopy = copy(array, size);

firstTask(arrayFirstCopy, maxIndex);

printf\_s("\narray for the first task: \n");

output(arrayFirstCopy, size);

freeArray(&arrayFirstCopy);

//Task 2

size\_t sizeForSecondArray = newSize(array, size, maxIndex);

int\* secondArray = getMemory(sizeForSecondArray);

secondTask(array, secondArray, size, maxIndex);

printf\_s("\narray for the second task:\n");

output(secondArray, sizeForSecondArray);

freeArray(&secondArray);

//Task 3

int\* arraySecondCopy = copy(array, size);

printf\_s("\ninput ammount of times that you want rule to work: ");

int rule = getInt();

thirdTask(arraySecondCopy, size, rule);

printf\_s("\narray for the third task:\n");

output(arraySecondCopy, size);

freeArray(&arraySecondCopy);

freeArray(&array);

return 0;

}

void firstTask(int\* array, const size\_t maxIndex)

{

array[maxIndex] = -1 \* array[maxIndex];

}

size\_t newSize(int\* const array, size\_t const size, const size\_t maxIndex)

{

size\_t temp = 0;

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

{

if (containOne(array[i]))

{

temp += 1;

}

}

return temp + size;

}

bool containOne(int value)

{

value = abs(value);

while (value > 0)

{

if (value % 10 == 1)

{

return true;

}

value /= 10;

}

return false;

}

void secondTask(int\* const array, int\* arrayCopy, size\_t const size, size\_t maxIndex)

{

size\_t j = 0;

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

{

if (containOne(array[i]))

{

arrayCopy[j] = array[i];

j++;

arrayCopy[j] = array[maxIndex];

}

else

{

arrayCopy[j] = array[i];

}

j++;

}

}

void thirdTask(int\* const array, const size\_t size, int rule)

{

if (rule > 0)

{

if (rule > size)

{

rule = size;

}

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

{

if (i % 2 == 0)

{

array[i] += i;

}

else

{

array[i] -= i;

}

}

}

}

void checkDomain(int const left, int const right)

{

if (left > right)

{

perror("wrong domain!!!!");

abort();

}

}

void checkBelongingDomain(int value, int const left, int const right)

{

if (value<left || value>right)

{

perror("value doesn't belong to domain!");

abort();

}

}

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

{

int\* arrayCopy = getMemory(size);

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

{

arrayCopy[i] = array[i];

}

return arrayCopy;

}

int getInt()

{

int entered\_number;

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

{

errno = EIO;

perror("Input Error");

abort();

}

return entered\_number;

}

size\_t getSize\_t()

{

int entered\_number = getInt();

if (entered\_number <= 0)

{

errno = ERANGE;

perror("Error: \n");

abort();

}

return (size\_t)entered\_number;

}

int\* getMemory(size\_t const size)

{

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

if (NULL == array)

{

perror("Error");

abort();

}

return array;

}

void keyboadArray(int\* const array, const size\_t size, int const left, int const right)

{

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

{

array[i] = getInt();

checkBelongingDomain(array[i], left, right);

}

}

void randomArray(int\* array, const size\_t size, int const left, int const right)

{

srand(time(NULL));

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

{

array[i] = rand() % (right - left + 1) + left;

}

}

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

{

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

{

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

}

}

void freeArray(int\*\* array)

{

free(\*array);

\*array = NULL;

}

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

{

size\_t j = 0;

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

{

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

{

j = i;

}

}

return j;

}

* 1. Решение различных тестовых примеров на C

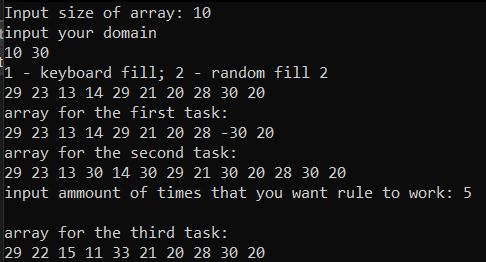


Рисунок  – Вывод программы 4.2 на C при корректном вводе значений с случайным заполнением



Рисунок  – Вывод программы на C при некорректном вводе размера массива

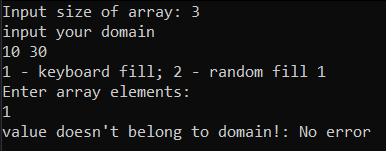


Рисунок  – Вывод программы 4.2 на C при выходе за пределы интервала при ручном вводе

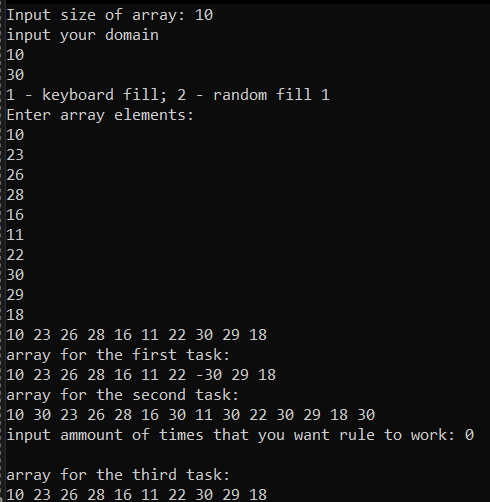


Рисунок  – Вывод программы 4.2 на C при корректном вводе значений с ручным заполнением массива

* 1. Подтверждение Approve 4.2

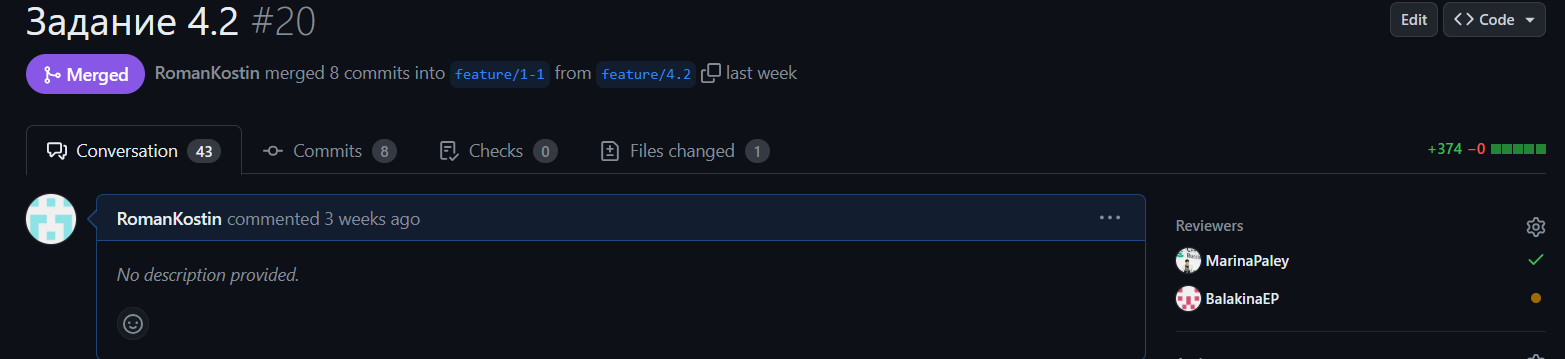


Рисунок  – Approve 4.2