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 12

Выполнил: ст. гр. ТКИ-142

Наседкин Тимофей Филиппович

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

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

Москва 2023

Оглавление

[1. Решение задачи 4-1 3](#_Toc156427109)

[1.1. Формулировка задачи 3](#_Toc156427110)

[1.2. Блок-схема алгоритма 4](#_Toc156427111)

[1.3. Код программы на языке C 12](#_Toc156427112)

[1.4. Результаты выполнения программы 18](#_Toc156427113)

[1.5. Отметка о выполнении задания в веб-хостинге системы контроля версий 19](#_Toc156427114)

[2. Решение задачи 4-2 20](#_Toc156427115)

[2.1. Формулировка задачи 20](#_Toc156427116)

[2.2. Блок-схема алгоритма 21](#_Toc156427117)

[2.3. Код программы на языке С 31](#_Toc156427118)

[2.4. Результаты выполнения программы 37](#_Toc156427119)

[2.5. Отметка о выполнении задания в веб-хостинге системы контроля версий 38](#_Toc156427120)

# Решение задачи 4-1

## Формулировка задачи

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 Условие задачи

|  |  |  |
| --- | --- | --- |
| Вариант | Задачи | Интервал |

|  |  |  |
| --- | --- | --- |
| 12 | 1. Найти произведение четных элементов, значения которых по модулю меньше 5. 2. Найти количество тех элементов, значения которых нечетны и по модулю превосходят заданное число А. 3. Умножить все нечетные элементы массива, кратные 3, на его номер. | [-10;20] |

## Блок-схема алгоритма

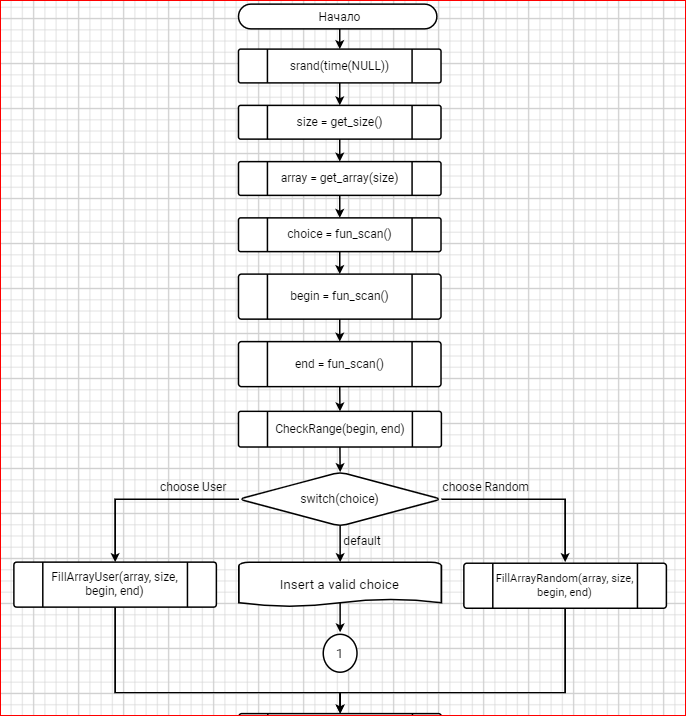
Блок-схема основного алгоритма представлена ниже (Рисунок 1), (Рисунок 2). Блок-схемы дополнительных функций представлены ниже (Рисунок 3) - (Рисунок 14). 

Рисунок 1‑1 Блок-схема основного алгоритма

*Рисунок STYLEREF 1 \s 0 SEQ Рисунок\_ \\* ARABIC \s 1 1 Блок-схема основного алгоритма.*

*Рисунок STYLEREF 1 \s 0 SEQ Рисунок\_ \\* ARABIC \s 1 2 Блок-схема основного алгоритма.*

*Рисунок STYLEREF 1 \s 0 SEQ Рисунок\_ \\* ARABIC \s 1 3 Блок-схема основного алгоритма.*

*Рисунок STYLEREF 1 \s 1 SEQ Рисунок \\* ARABIC \s 1 1 Блок-схема основного алгоритма.*

*Рисунок STYLEREF 1 \s 1 SEQ Рисунок \\* ARABIC \s 1 2*

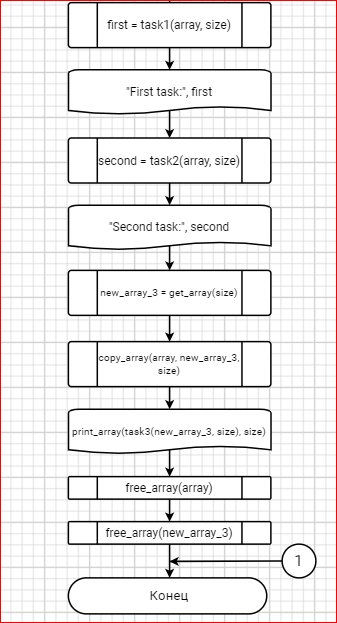


Рисунок 1‑2 Блок-схема основного алгоритма

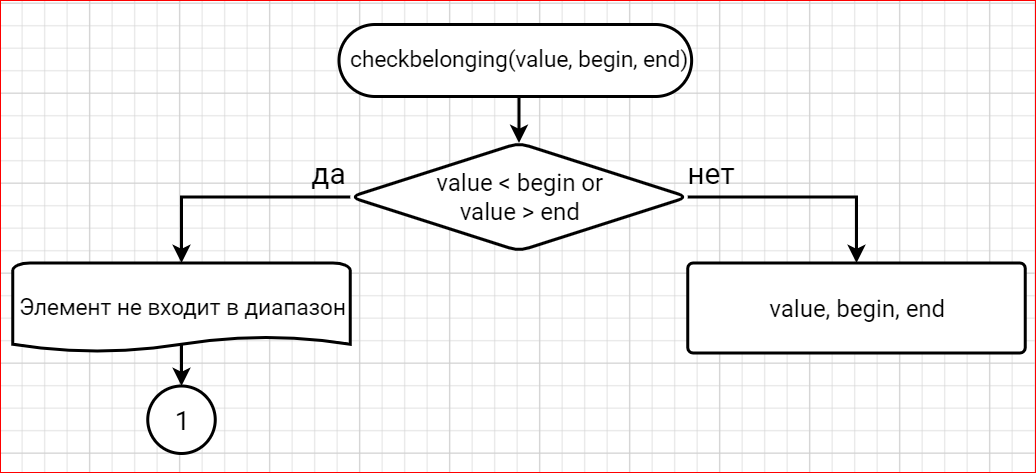


Рисунок 1‑3 Блок-схема функции checkbelonging().

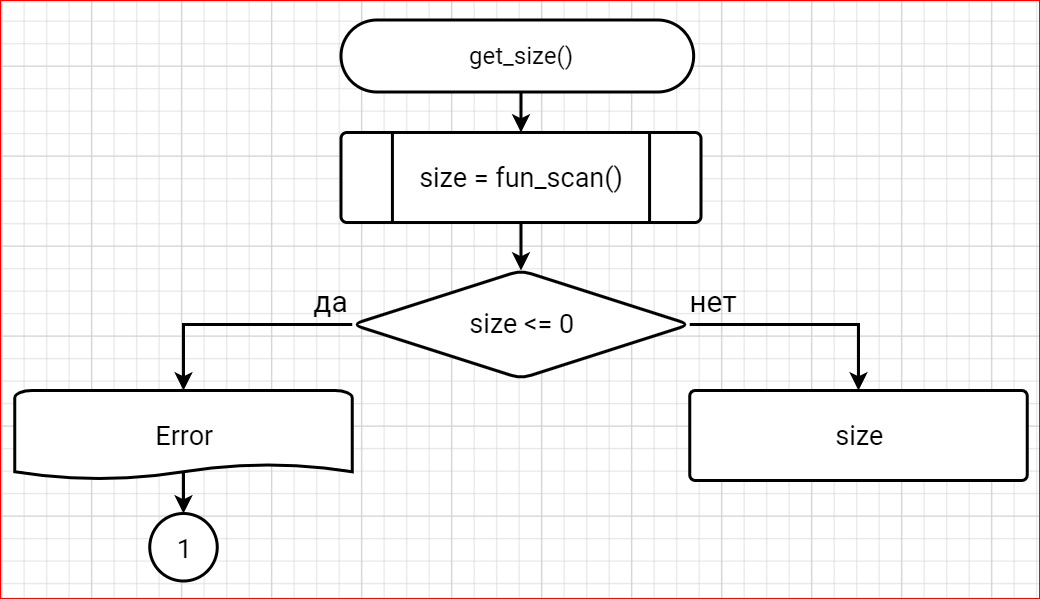


Рисунок 1‑4 Блок-схема функции get\_size().

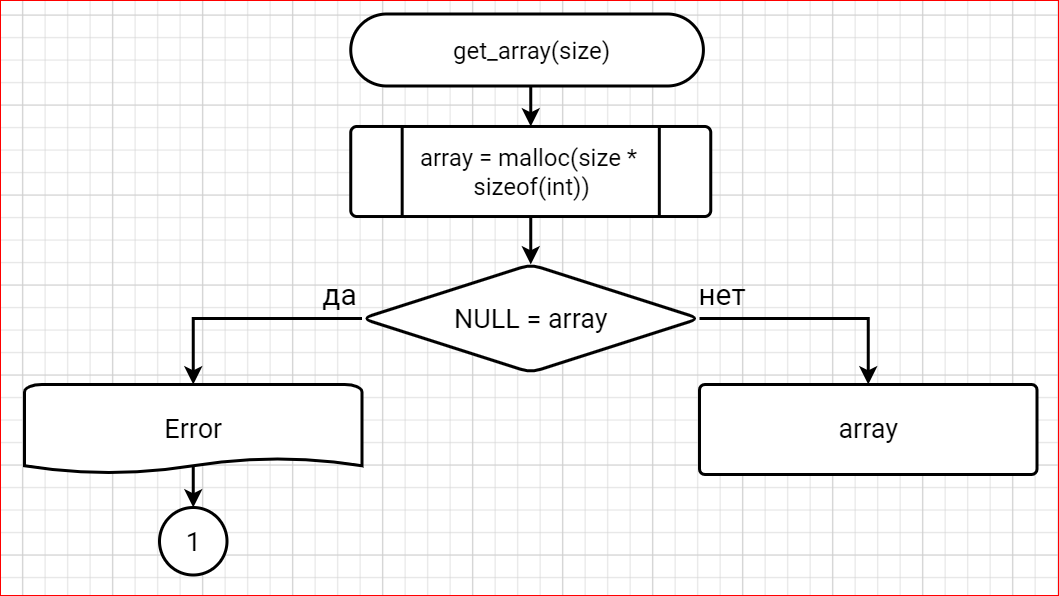


Рисунок 1‑5 Блок-схема функции get\_array().

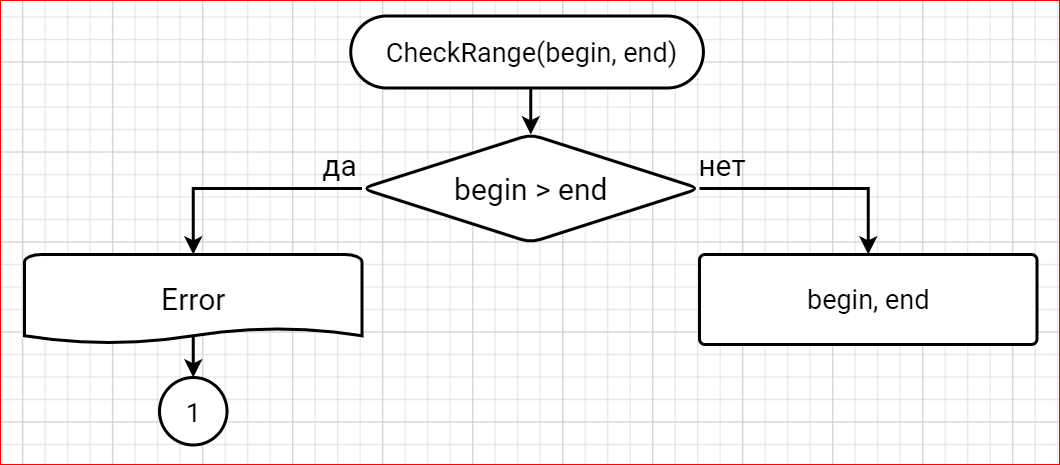


Рисунок 1‑6 Блок-схема функции CheckRange().

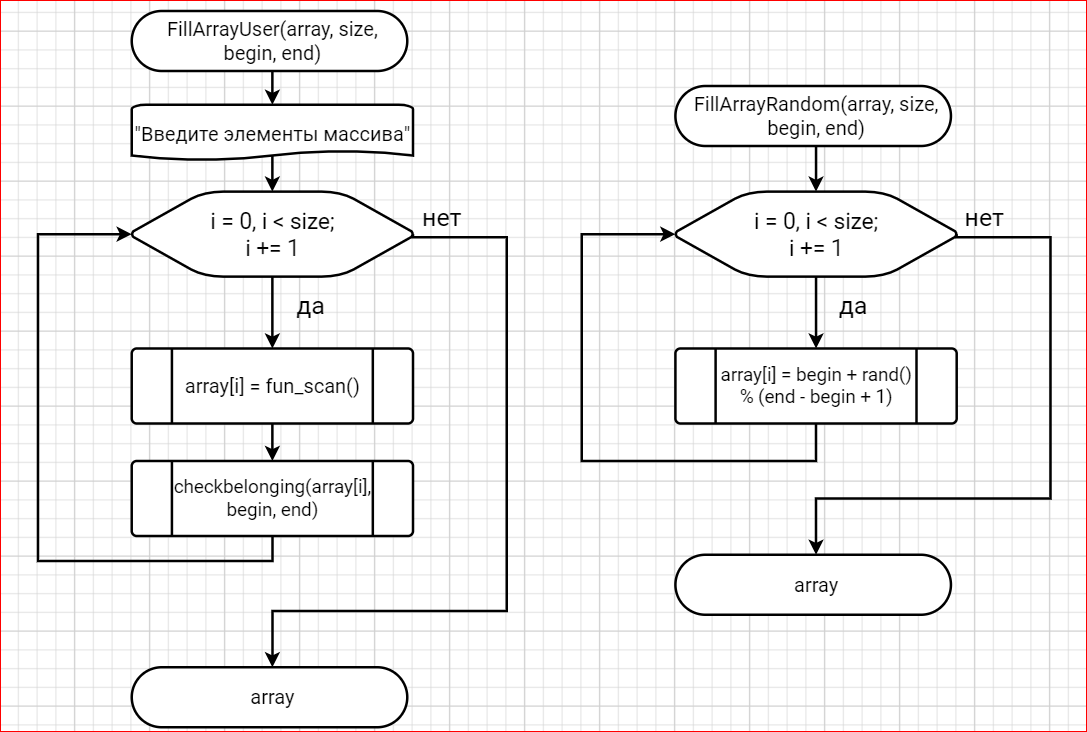


Рисунок 1‑7 Блок-схемы функций FillArrayUser(), FillArrayRandom().

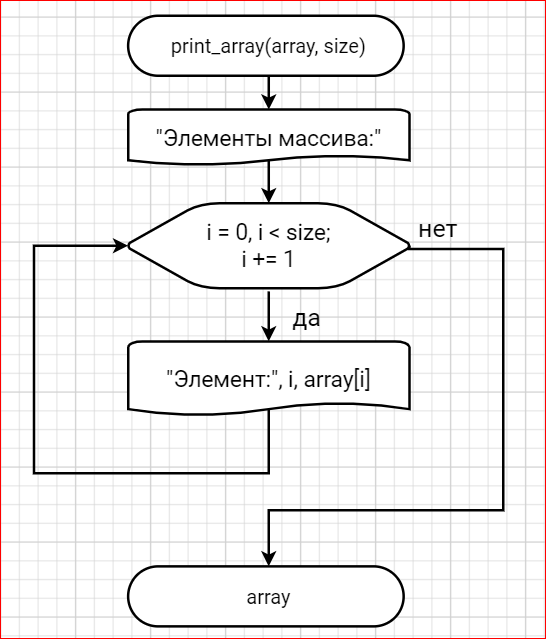


Рисунок 1‑8 Блок-схема функции print\_array().

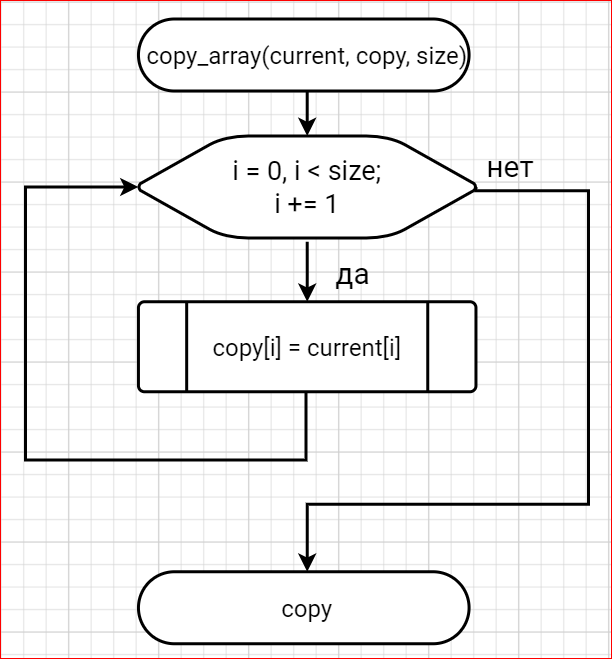


Рисунок 1‑9 Блок-схема функции copy\_array()

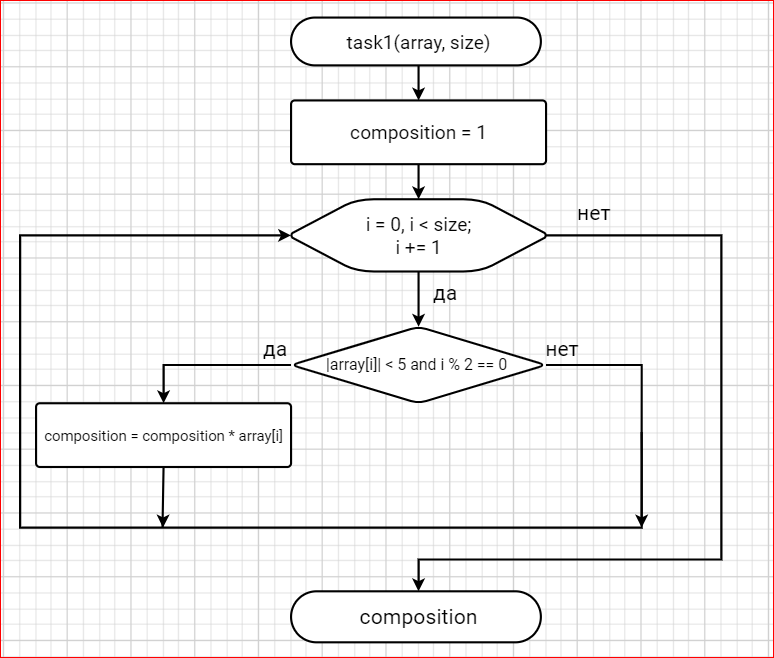
.

Рисунок 1‑10 Блок-схема функции task1().

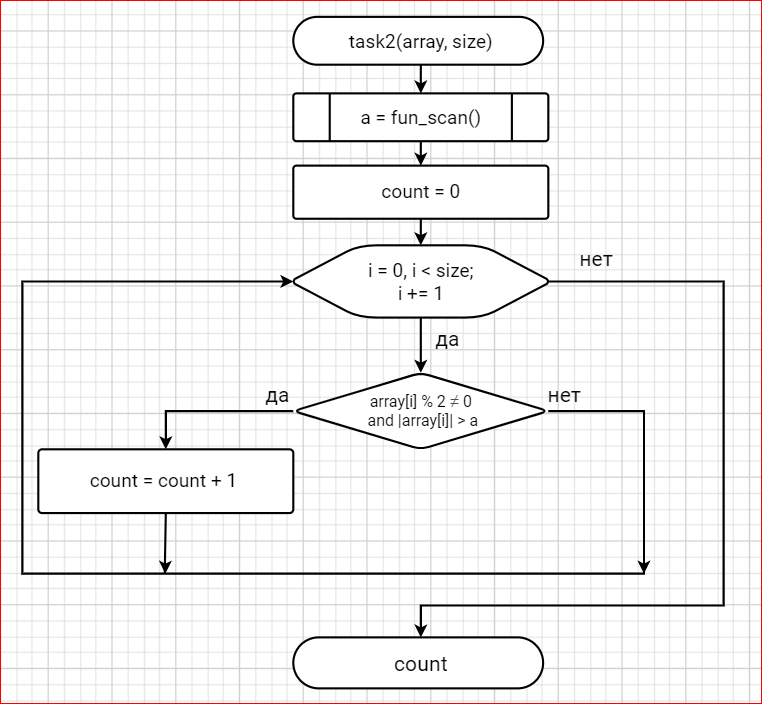


Рисунок 1‑11 Блок-схема функции task2()

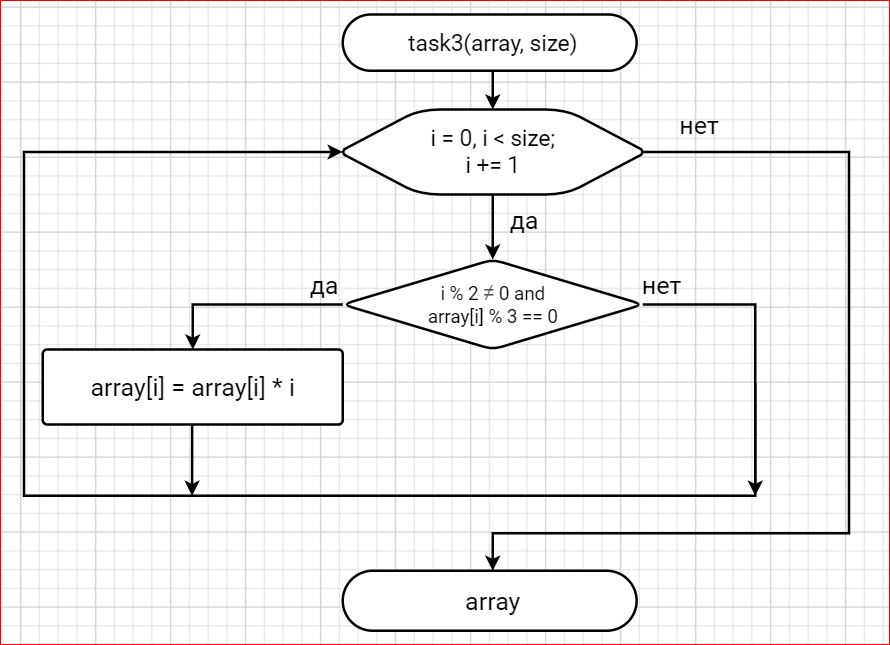
.

Рисунок 1‑12 Блок-схема функции task3().

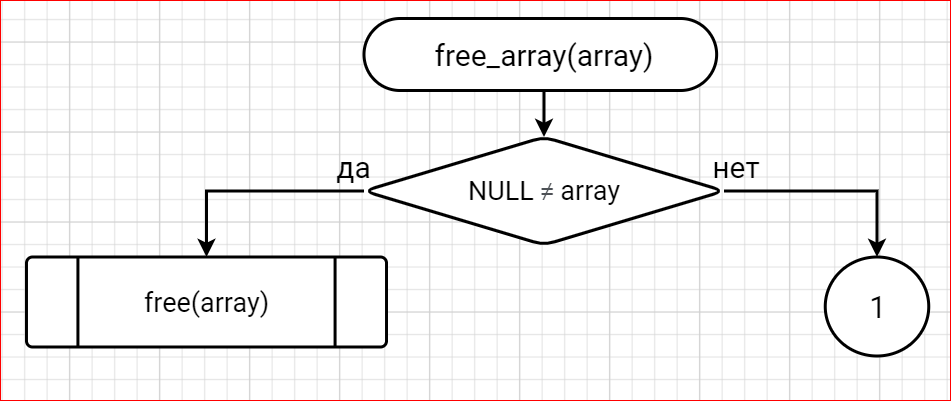


Рисунок 1‑13 Блок-схема функции free\_array().

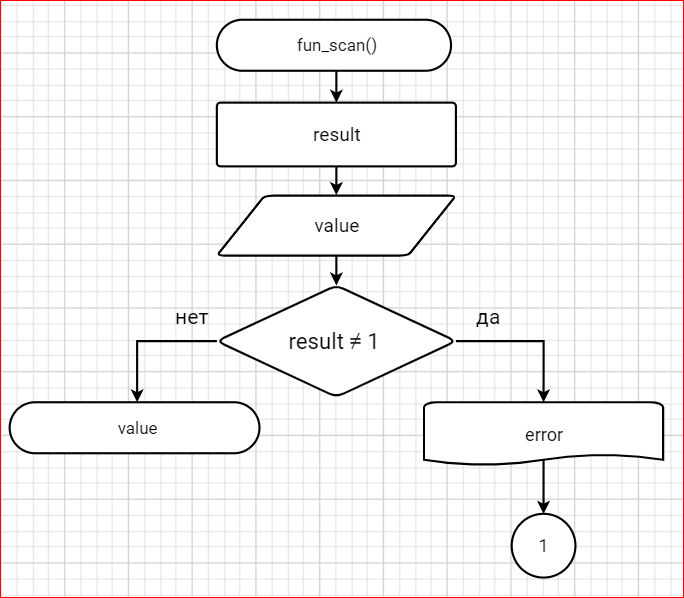


Рисунок 1‑14 Блок-схема функции fun\_scan().

## Код программы на языке C

#include <stdio.h>

#include <errno.h>

#include <stdlib.h>

#include <time.h>

#include <malloc.h>

#include <math.h>

#include <stdbool.h>

/\*\*

\* @brief Функция проверки принадлежности вводимого значения к диапазону

\* @param value вводимое значение

\* @param begin начало диапазона

\* @param end конец диапазона

\*/

void checkbelonging(const int value, const int begin, const int end);

/\*\*

\* @brief Функция выделяет память под массив

\* @param size длина массива

\* @return array указатель на пустой массив

\*/

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

/\*\*

\* @brief Функция заполняет массив элементами, которые вводит пользователь

\* @param array указатель на заполняемый массив

\* @param size длина массива

\*/

void FillArrayUser(int\* array, const size\_t size, const int begin, const int end);

/\*\*

\* @brief Функция заполняет массив рандомными числами в диапазоне [begin : end]

\* @param array указатель на заполняемый массив

\* @param size длина массива

\*/

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

/\*\*

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

\* @return number положительное число

\*/

size\_t get\_size();

/\*\*

\* @brief Функция выводит массив на экран

\* @param array указатель на заполняемый массив

\* @param size длина массива

\*/

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

/\*\*

\* @brief Функция копирующая элементы одного массива в другой

\* @param Current исходный массив

\* @param Copy пустой массив

\* @param size длина массива

\*/

void copy\_array(int\* const current, int\* copy, const size\_t size);

/\*\*

\* @brief Функция нахождения произведения чётных элементов, значения которых по модулю меньше 5

\* @param array указатель на массив

\* @param size размерность массива

\* @return composition произведение элементов массива, которые удовлетворяют условию задачи

\*/

int task1(int\* const array, const size\_t size);

/\*\*

\* @brief Фунция нахождения количества элементов, значения которых нечетны и по модулю превосходят заданное А

\* @param array указатель на массив

\* @param size размерность массива

\* @return count количество элементов, удовлетворяющих условию задачи

\*/

int task2(int\* const array, const size\_t size);

/\*\*

\* @brief Функция умножения всех чётных элементов массива, кратных 3, на их номера

\* @param array указатель на массив

\* @param size размерность массива

\* @return Массив, после выполнения поставленных в условии задач

\*/

int \*task3(int\* array, const size\_t size);

/\*\*

\* @brief Функция, освобождающая массив

\* @param array указатель на массив

\*/

void free\_array(int\* array);

/\*\*

\* @brief Функция проверяет концы диапазона

\* @param begin начало диапазона

\* @param end конец диапазона

\*/

void CheckRange(const int begin, const int end);

/\*\*

\* @brief Функция присваивает целочисленное значение переменной

\* @param сообщение для пользователя

\* @return целочисленная цифра

\*/

int fun\_scan(char\* const message);

/\*\*

\* @brief Функция проверки ввода на число

\* @return Результат проверки

\*/

int check\_fun();

/\*\*

\* @brief структура хранит константы, указывающие выбор пользователя заполнения массива

\* @brief User хранит значение, вызывающее ввод массива вручную

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

\*/

enum Choices

{

User = 1,

Random = 2

};

/\*\*

\* @brief Точка входа в программу

\* @return Вывод 1, если программа работает неверно. Иначе вывод 0

\*/

int main()

{

srand(time(NULL));

size\_t size = get\_size();

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

int choice = fun\_scan("Выберите способ заполнения массива\n 1. Вручную, 2. Автозаполнение\n");

const int begin = fun\_scan("Введите нижнюю границу диапазона: "), end = fun\_scan("Введите верхнюю границу диапазона: ");

CheckRange(begin, end);

switch ((enum Choices)choice)

{

case User:

FillArrayUser(Array, size, begin, end);

break;

case Random:

FillArrayRandom(Array, size, begin, end);

break;

default:

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

return 1;

}

printf("First task: %d\n", task1(Array, size));

printf("Second task: %d\n", task2(Array, size));

printf("Third task: ");

int\* new\_array\_3 = get\_array(size);

copy\_array(Array, new\_array\_3, size);

print\_array(task3(new\_array\_3, size), size);

free\_array(Array);

free\_array(new\_array\_3);

return 0;

}

void checkbelonging(const int value, const int begin, const int end)

{

if ((value < begin) || (value > end))

{

printf("Элемент не входит в заданный диапазон");

abort();

}

}

size\_t get\_size()

{

int size = fun\_scan("Введите размер массива: ");

if (size <= 0)

{

errno = ERANGE;

perror("Error :");

abort();

}

return (size\_t)size;

}

int\* get\_array(const int size)

{

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

if (NULL == array)

{

errno = ENOMEM;

perror("Error :");

abort();

}

return array;

}

void CheckRange(const int begin, const int end)

{

if (begin > end)

{

puts("error ");

abort();

}

}

void FillArrayUser(int\* array, const size\_t size, const int begin, const int end)

{

puts("Введите элементы массива: ");

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

{

array[i] = fun\_scan("Введите элемент = ");

checkbelonging(array[i], begin, end);

}

}

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

{

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

{

array[i] = begin + rand() % (end - begin + 1);

}

}

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

{

puts("Элементы массива: ");

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

{

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

}

}

void copy\_array(int\* const current, int\* copy, const size\_t size)

{

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

{

copy[i] = current[i];

}

}

int task1(int\* const array, const size\_t size)

{

int composition = 1;

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

{

if ((abs(array[i]) < 5) && (i % 2 == 0))

{

composition \*= array[i];

}

}

return composition;

}

int task2(int\* const array, const size\_t size)

{

int a = check\_fun();

int count = 0;

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

{

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

{

count += 1;

}

}

return count;

}

int \*task3(int\* array, const size\_t size)

{

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

{

if ((i % 2 != 0) && (array[i] % 3 == 0))

{

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

}

}

return array;

}

void free\_array(int\* array)

{

if (NULL != array)

{

free(array);

}

}

int fun\_scan(char\* const message)

{

int value = 0;

printf("%s", message);

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

if (result != 1)

{

errno = EIO;

perror("Error :");

abort();

}

return value;

}

int check\_fun()

{

int value;

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

if (result != 1)

{

puts("error");

abort();

}

return value;

}

## Результаты выполнения программы

Результаты выполнения программы представлены ниже (Рисунок 15).

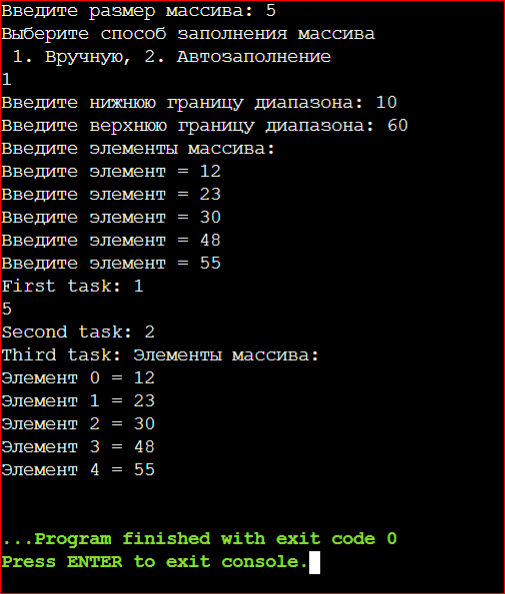


Рисунок 1‑15 Результат выполнения программы.

## Отметка о выполнении задания в веб-хостинге системы контроля версий

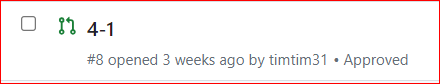


Рисунок 1‑16 Отметка о выполнении.

# Решение задачи 4-2

## Формулировка задачи

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 |

|  |  |  |
| --- | --- | --- |
| 12 | 1. Заменить последний отрицательный элемент массива на модуль первого элемента массива. 2. Удалить из него все элементы, у которых первая и вторая цифры одинаковые. 3. Из элементов массива P сформировать массив M той же размерности по правилу: если элемент четный, то Mi=i\*Pi , если нечетный, то Mi=-Pi. | [-10;20] |

## Блок-схема алгоритма

Блок-схема основного алгоритма представлена ниже (Рисунок 1), (Рисунок 2), (Рисунок 3). Блок-схемы дополнительных функций представлены ниже (Рисунок 4) – (Рисунок 16).

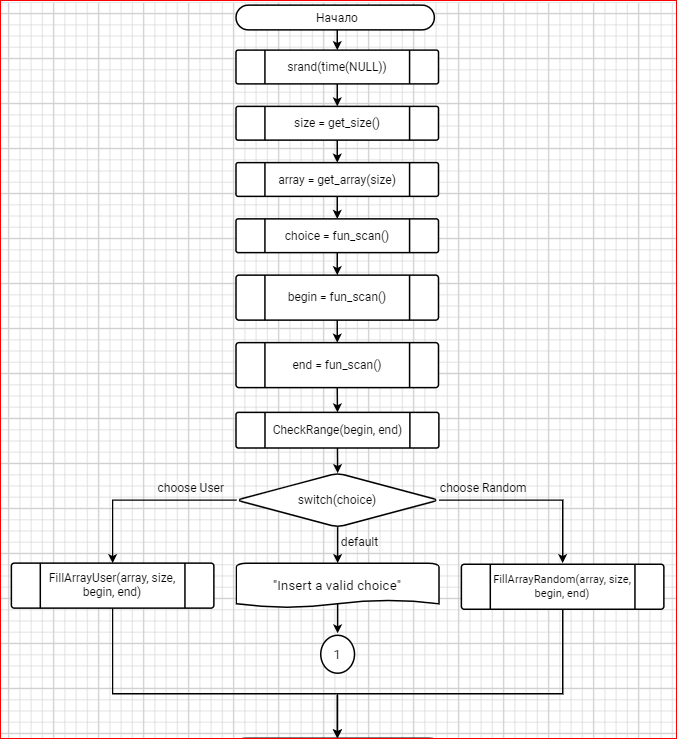


Рисунок 2‑1 Блок-схема основного алгоритма

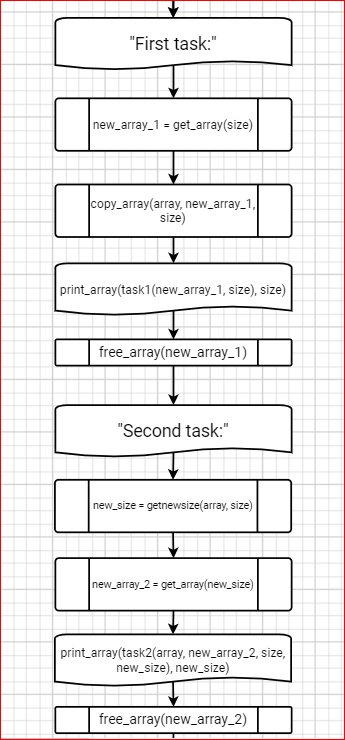


Рисунок 2‑2 Блок-схема основного алгоритма

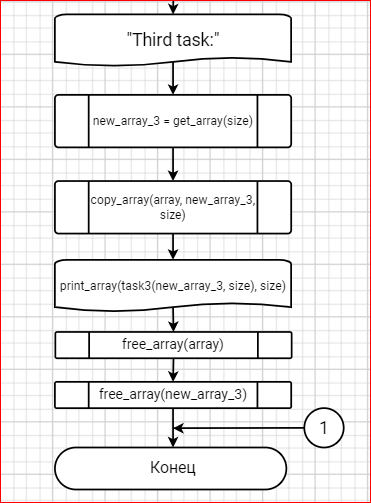


Рисунок 2‑3 Блок-схема основного алгоритма

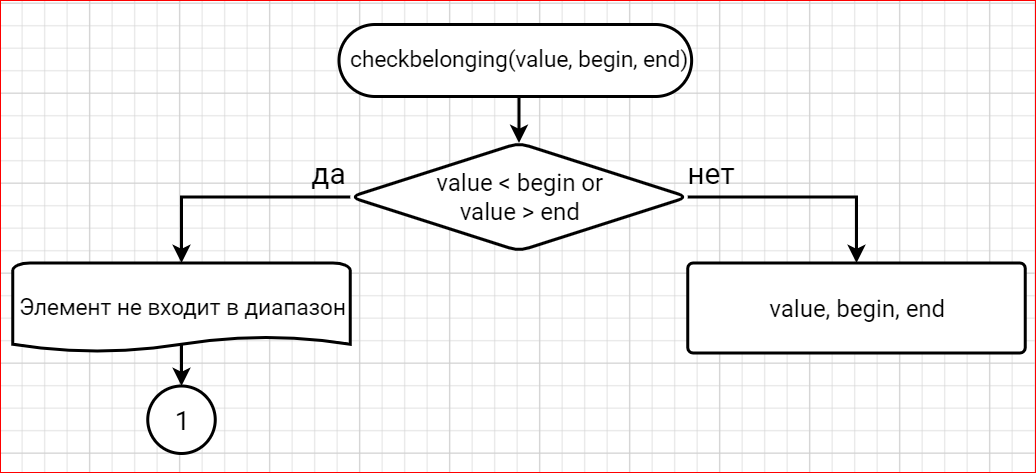


Рисунок 2‑4 Блок-схема функции checkbelonging().

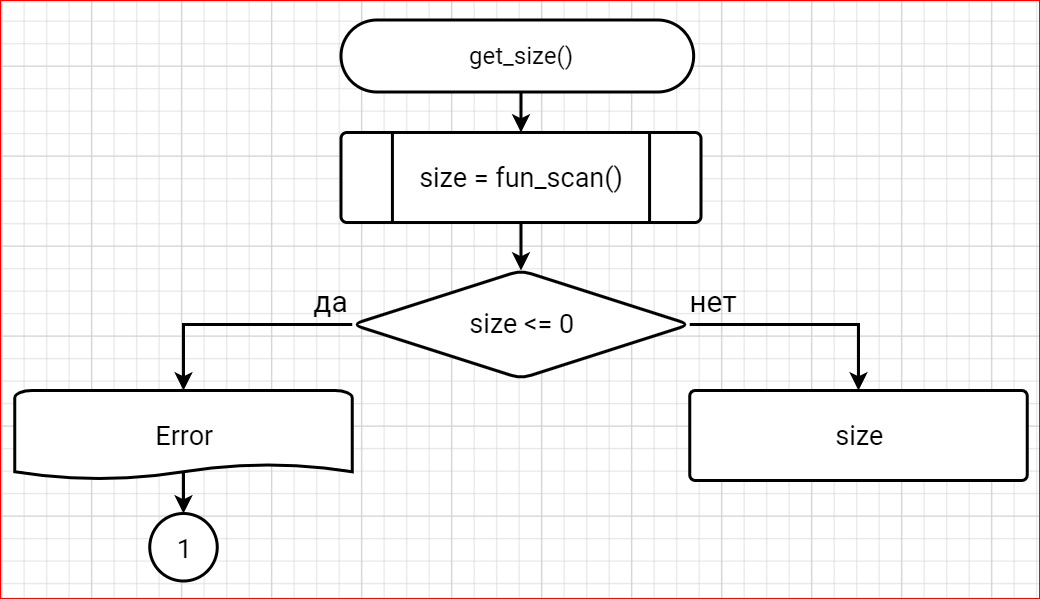


Рисунок 2‑5 Блок-схема функции get\_size().

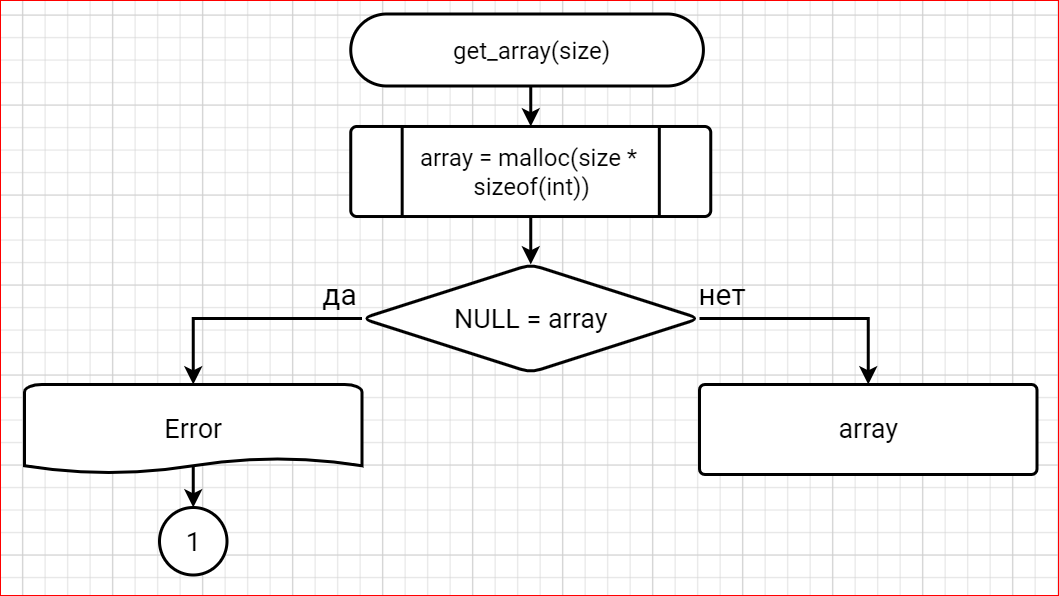


Рисунок 2‑6 Блок-схема функции get\_array().

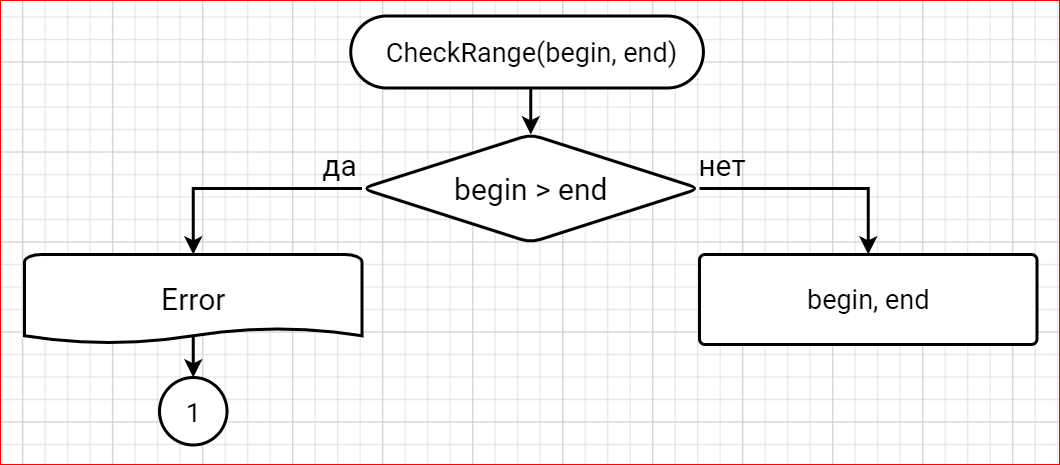


Рисунок 2‑7 Блок-схема функции CheckRange().

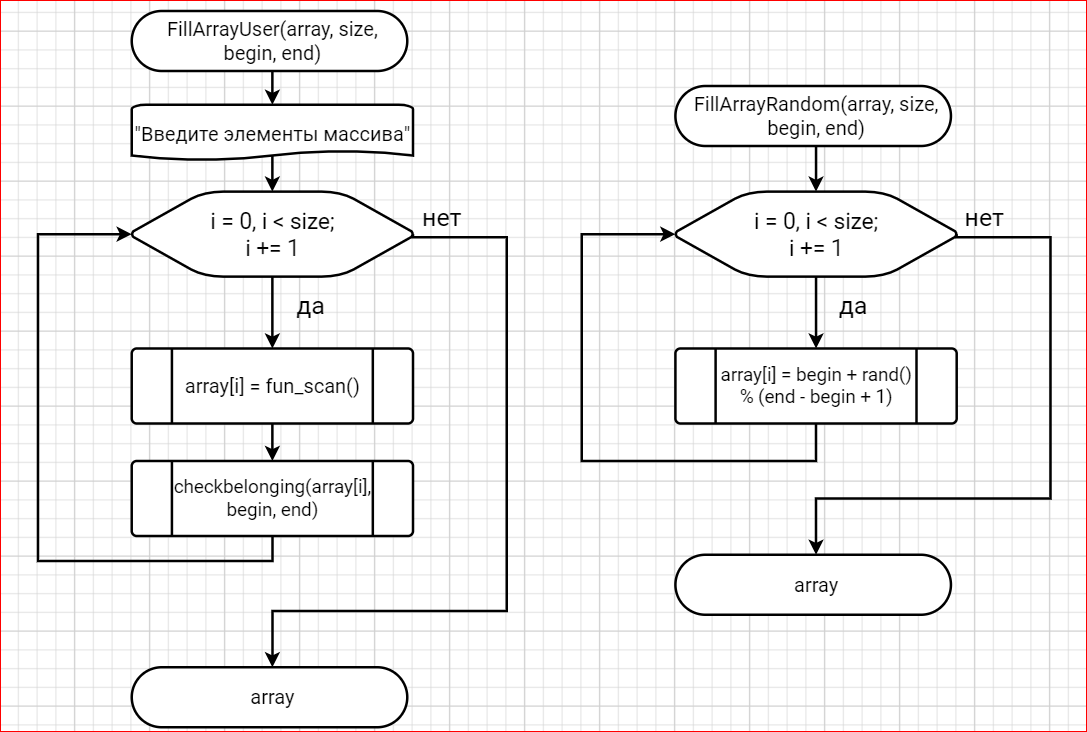


Рисунок 2‑8 Блок-схемы функций FillArrayUser(), FillArrayRandom().

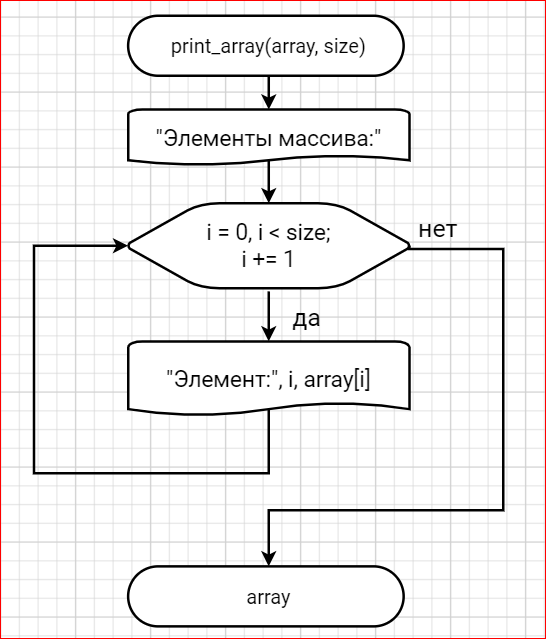


Рисунок 2‑9 Блок-схема функции print\_array().

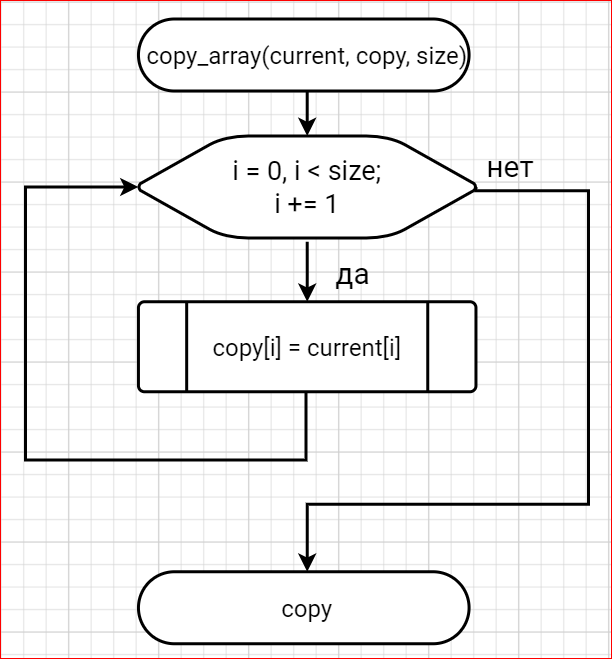


Рисунок 2‑10 Блок-схема функции copy\_array().

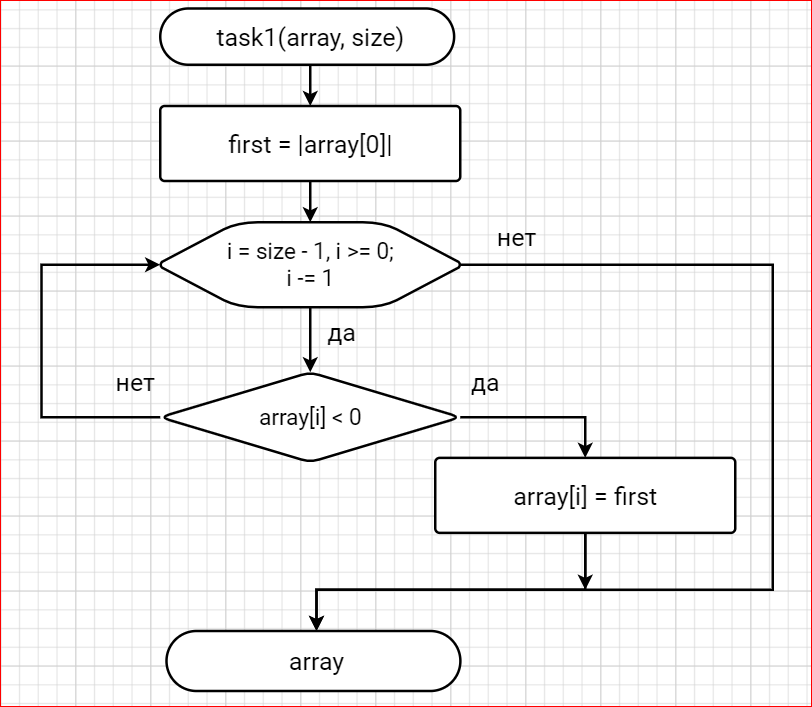


Рисунок 2‑11 Блок-схема функции task1().

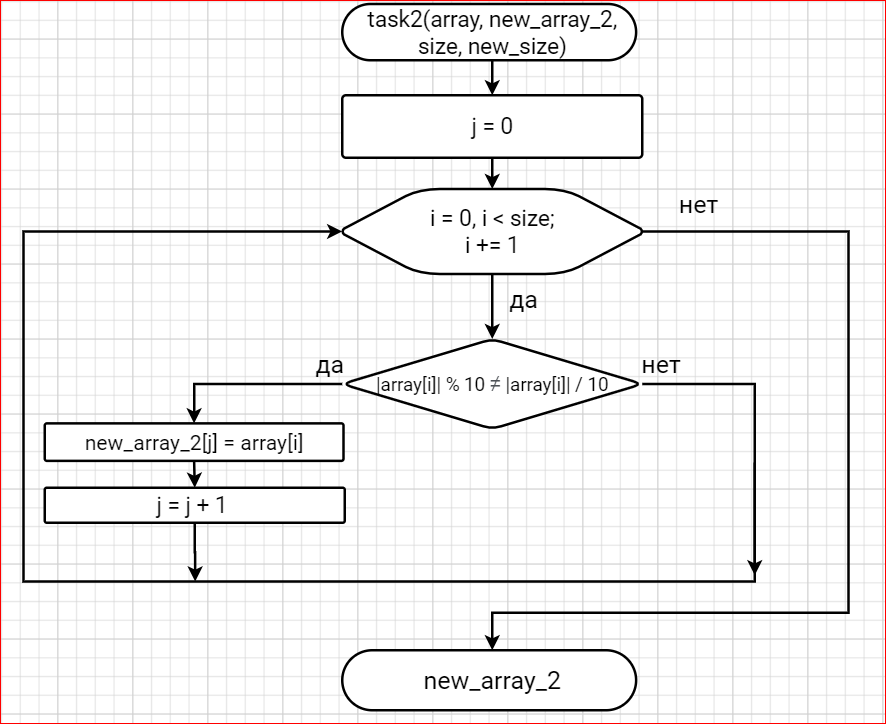


Рисунок 2‑12 Блок-схема функции task2().

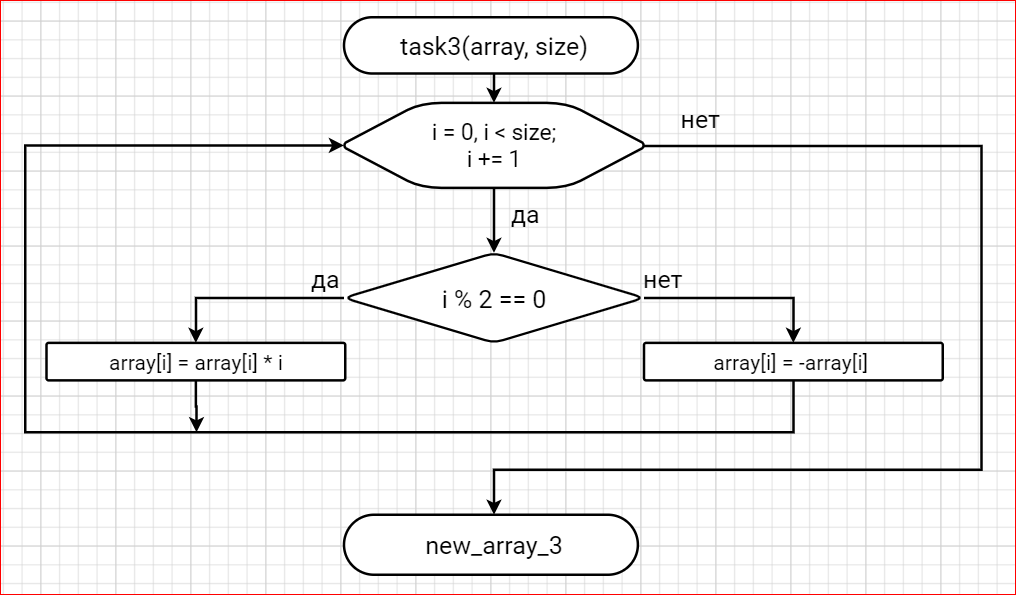


Рисунок 2‑13 Блок-схема функции task3().

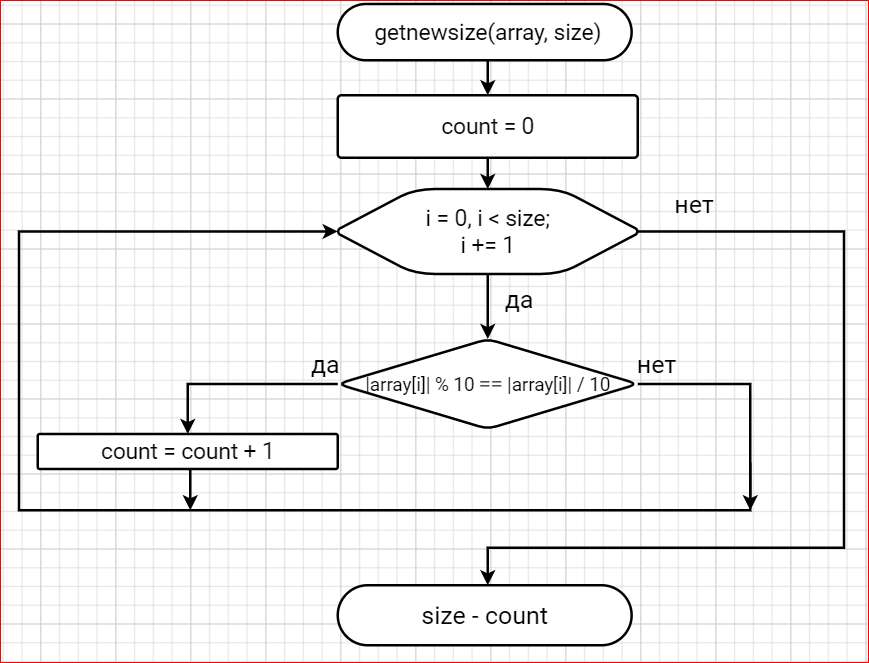


Рисунок 2‑14 Блок-схема функции getnewsize().

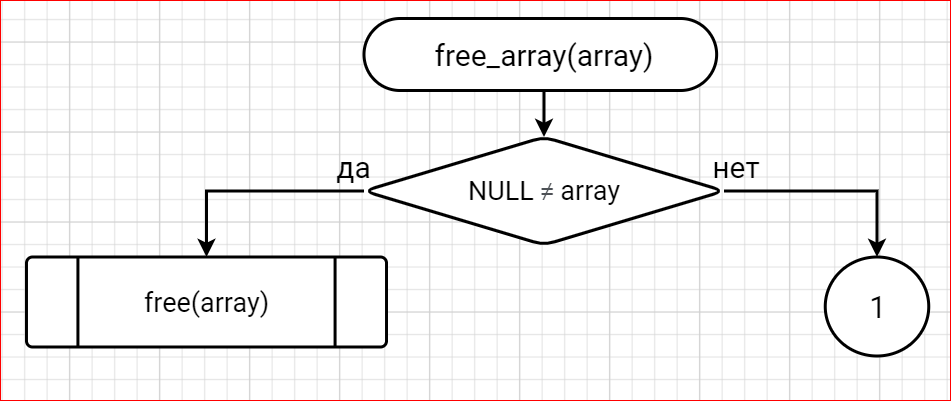


Рисунок 2‑15 Блок-схема функции free\_array().

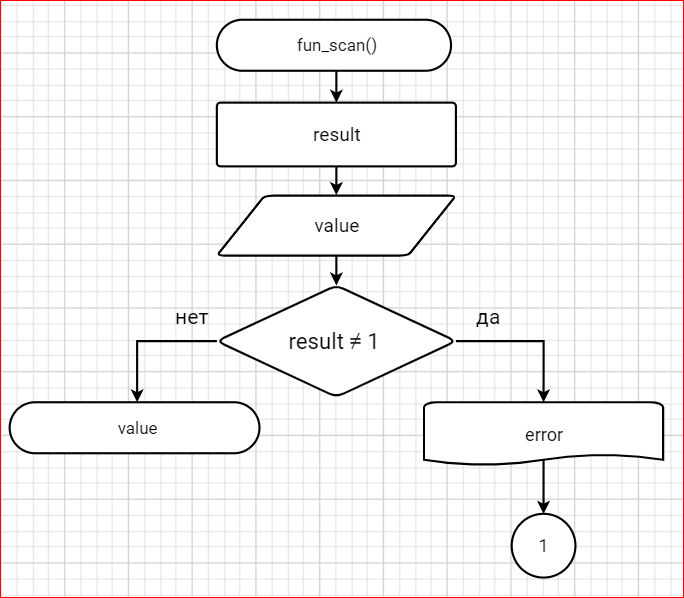


Рисунок 2‑16 Блок-схема функции fun\_scan().

## Код программы на языке С

#include <stdio.h>

#include <errno.h>

#include <stdlib.h>

#include <time.h>

#include <malloc.h>

/\*\*

\* @brief Функция проверки принадлежности вводимого значения к диапазону

\* @param value вводимое значение

\* @param begin начало диапазона

\* @param end конец диапазона

\*/

void checkbelonging(const int value, const int begin, const int end);

/\*\*

\* @brief Функция выделяет память под массив

\* @param size длина массива

\* @return array указатель на пустой массив

\*/

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

/\*\*

\* @brief Функция заполняет массив элементами, которые вводит пользователь

\* @param array указатель на заполняемый массив

\* @param size длина массива

\*/

void FillArrayUser(int\* array, const size\_t size, const int begin, const int end);

/\*\*

\* @brief Функция заполняет массив рандомными числами в диапазоне [begin : end]

\* @param array указатель на заполняемый массив

\* @param size длина массива

\*/

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

/\*\*

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

\* @return number положительное число

\*/

size\_t get\_size();

/\*\*

\* @brief Функция выводит массив на экран

\* @param array указатель на заполняемый массив

\* @param size длина массива

\*/

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

/\*\*

\* @brief Функция копирующая элементы одного массива в другой

\* @param Current исходный массив

\* @param Copy пустой массив

\* @param size длина массива

\*/

void copy\_array(int\* const current, int\* copy, const size\_t size);

/\*\*

\* @brief Функция замены последнего отрицательного элемента массива на модуль первого

\* @param array указатель на массив

\* @param size размерность массива

\* @return Массив, после выполнения поставленных в условии задач

\*/

int \*task1(int\* array, const size\_t size);

/\*\*

\* @brief Фунция нахождения размера нового массива для задания task2

\* @param array указатель на массив

\* @param size размерность массива

\* @return Размер нового массива

\*/

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

/\*\*

\* @brief Фунция удаления из массива элементов, у которых первая и вторая цифры одинаковые

\* @param array указатель на массив

\* @param size размерность массива

\* @return Массив, после выполнения поставленных в условии задач

\*/

int \*task2(const int\* array, int\* new\_array\_2, const size\_t size, const size\_t new\_size);

/\*\*

\* @brief Функция формирования нового массива M по правилу: элемент четный, тогда Mi = i \* Pi

\* @brief Если нечетный, тогда Mi = -Pi

\* @param array указатель на массив

\* @param size размерность массива

\* @return Массив, после выполнения поставленных в условии задач

\*/

int \*task3(int\* array, const size\_t size);

/\*\*

\* @brief Функция, освобождающая массив

\* @param array указатель на массив

\*/

void free\_array(int\* array);

/\*\*

\* @brief Функция проверяет концы диапазона

\* @param begin начало диапазона

\* @param end конец диапазона

\*/

void CheckRange(const int begin, const int end);

/\*\*

\* @brief Функция присваивает целочисленное значение переменной

\* @param сообщение для пользователя

\* @return целочисленная цифра

\*/

int fun\_scan(char\* const message);

/\*\*

\* @brief структура хранит константы, указывающие выбор пользователя заполнения массива

\* @brief User хранит значение, вызывающее ввод массива вручную

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

\*/

enum Choices

{

User = 1,

Random = 2

};

/\*\*

\* @brief Точка входа в программу

\* @return Вывод 0, если программа работает неверно. Иначе вывод 1

\*/

int main()

{

srand(time(NULL));

size\_t size = get\_size();

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

int choice = fun\_scan("Выберите способ заполнения массива\n 1. Вручную, 2. Автозаполнение\n");

const int begin = fun\_scan("Введите нижнюю границу диапазона: "), end = fun\_scan("Введите верхнюю границу диапазона: ");

CheckRange(begin, end);

switch ((enum Choices)choice)

{

case User:

FillArrayUser(array, size, begin, end);

break;

case Random:

FillArrayRandom(array, size, begin, end);

break;

default:

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

return 1;

}

printf("First task: ");

int\* new\_array\_1 = get\_array(size);

copy\_array(array, new\_array\_1, size);

print\_array(task1(new\_array\_1, size), size);

free\_array(new\_array\_1);

printf("Second task: ");

int new\_size = getnewsize(array, size);

int\* new\_array\_2 = get\_array(new\_size);

print\_array(task2(array, new\_array\_2, size, new\_size), new\_size);

free\_array(new\_array\_2);

printf("Third task: ");

int\* new\_array\_3 = get\_array(size);

copy\_array(array, new\_array\_3, size);

print\_array(task3(new\_array\_3, size), size);

free\_array(array);

free\_array(new\_array\_3);

return 0;

}

void checkbelonging(const int value, const int begin, const int end)

{

if ((value < begin) || (value > end))

{

printf("Элемент не входит в заданный диапазон");

abort();

}

}

size\_t get\_size()

{

int size = fun\_scan("Введите размер массива: ");

if (size <= 0)

{

errno = ERANGE;

perror("Error :");

abort();

}

return (size\_t)size;

}

int\* get\_array(const int size)

{

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

if (NULL == array)

{

errno = ENOMEM;

perror("Error :");

abort();

}

return array;

}

void CheckRange(const int begin, const int end)

{

if (begin > end)

{

puts("error ");

abort();

}

}

void FillArrayUser(int\* array, const size\_t size, const int begin, const int end)

{

puts("Введите элементы массива: ");

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

{

array[i] = fun\_scan("Введите элемент = ");

checkbelonging(array[i], begin, end);

}

}

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

{

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

{

array[i] = begin + rand() % (end - begin + 1);

}

}

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

{

puts("Элементы массива: ");

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

{

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

}

}

void copy\_array(int\* const current, int\* copy, const size\_t size)

{

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

{

copy[i] = current[i];

}

}

int \*task1(int\* array, const size\_t size)

{

int first = abs(array[0]);

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

{

if (array[i] < 0)

{

array[i] = first;

break;

}

}

return array;

}

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

{

int count = 0;

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

{

if ((abs(array[i]) % 10) == (abs(array[i]) / 10))

{

count += 1;

}

}

return(size - count);

}

int \*task2(const int\* array, int\* new\_array\_2, const size\_t size, const size\_t new\_size)

{

int j = 0;

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

{

if ((abs(array[i]) % 10) != (abs(array[i]) / 10))

{

new\_array\_2[j] = array[i];

j+=1;

}

}

return(new\_array\_2);

}

int \*task3(int\* array, const size\_t size)

{

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

{

if (i % 2 == 0)

{

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

}

else

{

array[i] = -array[i];

}

}

return array;

}

void free\_array(int\* array)

{

if (NULL != array)

{

free(array);

}

}

int fun\_scan(char\* const message)

{

int value = 0;

printf("%s", message);

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

if (result != 1)

{

errno = EIO;

perror("Error :");

abort();

}

return value;

}

## Результаты выполнения программы

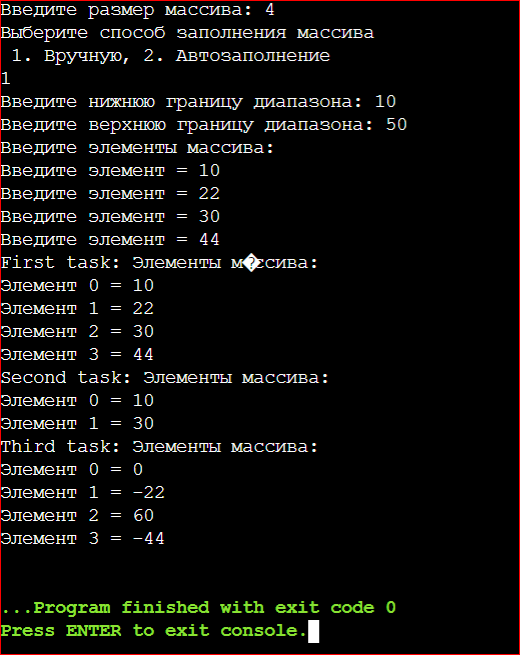
Результаты выполнения программы представлены ниже (Рисунок 17)

Рисунок 2‑17 РЕЗУЛЬТАТ ВЫПОЛНЕНИЯ ПРОГРАММЫ.

## Отметка о выполнении задания в веб-хостинге системы контроля версий

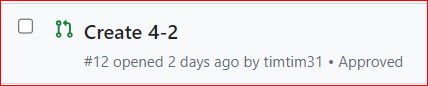


Рисунок 2‑18 ОТМЕТКА О ВЫПОЛНЕНИИ.