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 №4.3

По дисциплине «Языки программирования»

ВАРИАНТ 4

Выполнил: ст. гр. ТКИ-141

Бышовец Михаил Александрович

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

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

Москва 2024

1. Формулировка задания

Создать многомерный массив nˣm из *n* целых чисел. Реализовать возможность заполнения массива, как случайными числами, так и с помощью клавиатуры по желанию пользователя. Пункты задания организовать в виде функций (методов). Вывести массив на экран.

Таблица 1 – Исходные данные

|  |  |
| --- | --- |
| Вариант | Задачи |
| 4 | 1. Заменить минимальный по модулю элемент каждого столбца на противоположный. 2. Удалить все строки, содержащие максимальные элементы. |

1. Блок-схема алгоритма

Блок-схема основного алгоритма представлена ниже (Рисунок 1). Блок-схема функций input (Рисунок 2), initialize\_list (Рисунок 3), initialize (Рисунок 4), manually\_list (Рисунок 5), manually\_init и random\_init (Рисунок 6), random\_list (Рисунок 7), функции is\_positive (Рисунок 8) и функции, positive\_input (Рисунок 9), check(Рисунок 10), check\_list(Рисунок 11), replace\_min\_element (Рисунок 12), min\_element (Рисунок 13), index\_max\_element (Рисунок 14), free\_all (Рисунок 15), remove\_list (Рисунок 16), move\_list (Рисунок 17), new\_list (рисунок 18) представлены ниже.

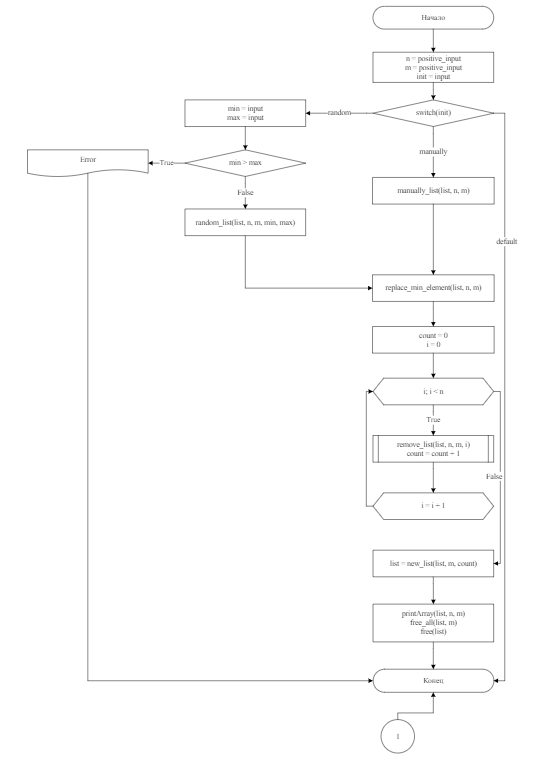


Рисунок 1 ­ Блок-схема основного алгоритма

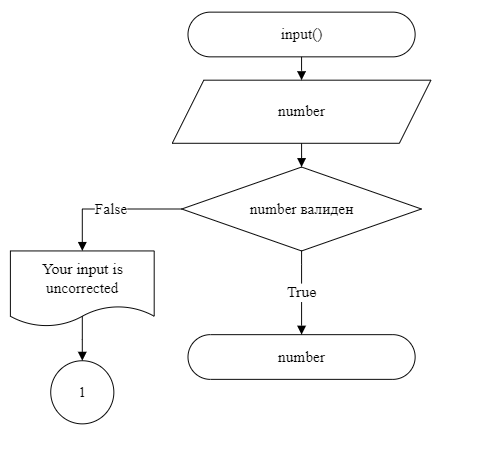


Рисунок 2 – Блок-схемf используемой функции input

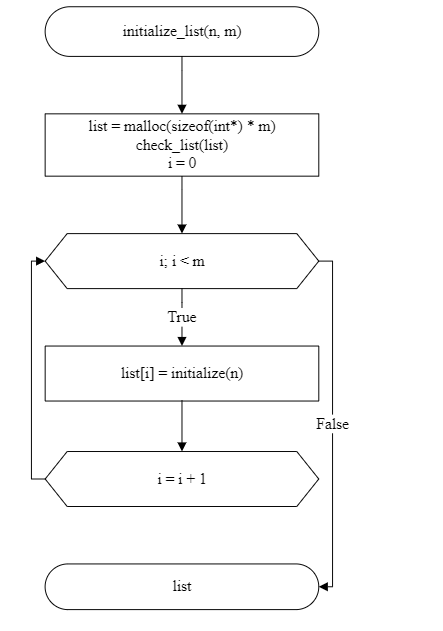


Рисунок 3 –  Блок-схема используемой функции initialize\_list

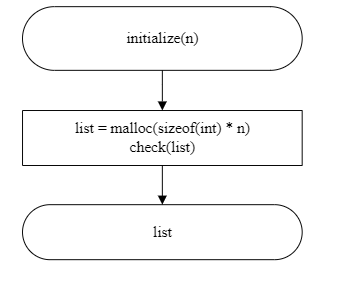


Рисунок 4 - Блок-схема используемой функции initialize

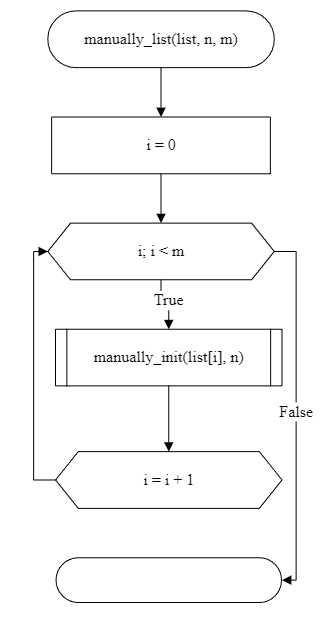


Рисунок 5 - Блок-схема используемой функции manually\_list

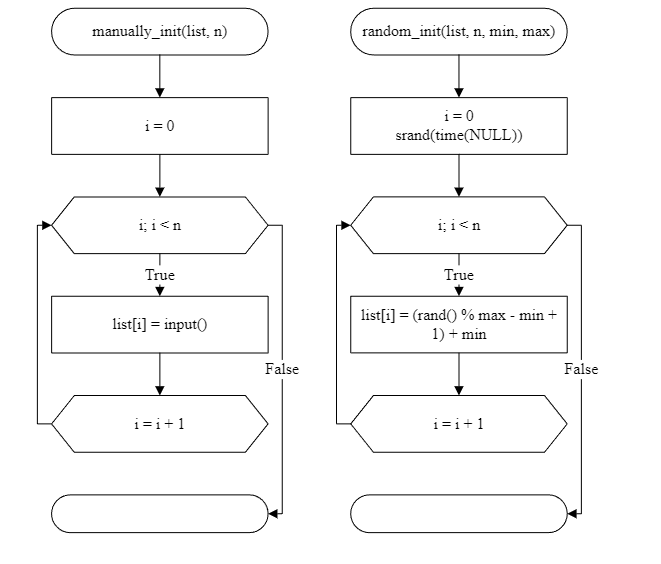


Рисунок 6 - Блок-схема используемых функий manually\_init и random\_init

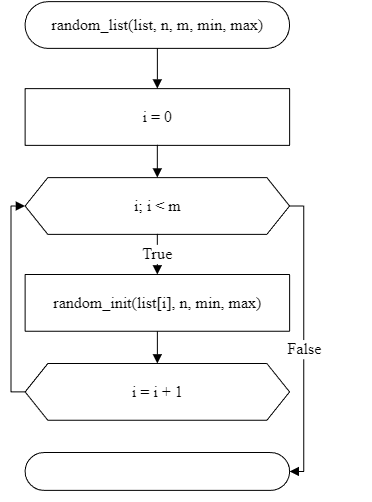


Рисунок 7 - используемые функции ввода positive\_input и input

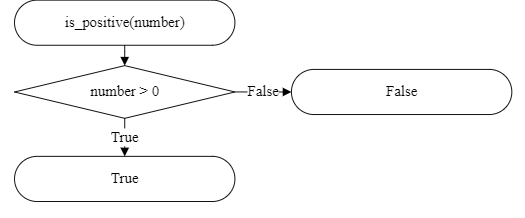


Рисунок 8 - Блок-схема используемой функции is\_positive

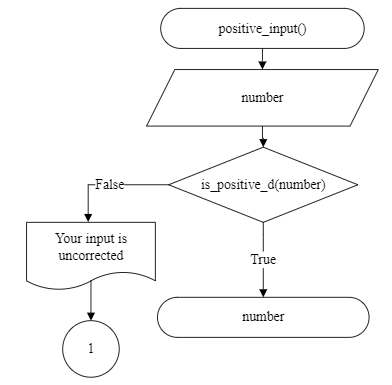


Рисунок 9 - Блок-схема используемой функции positive\_input

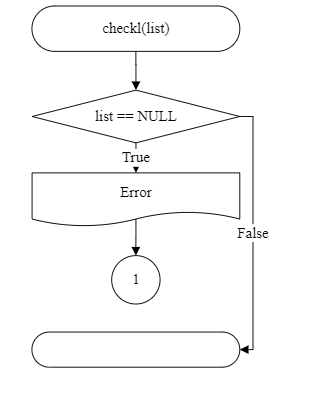


Рисунок 10 - Блок-схема используемой функции check

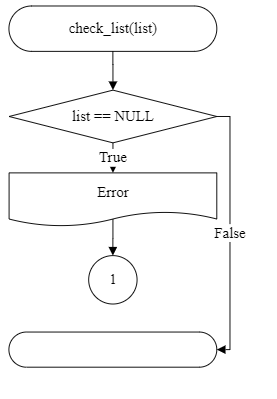


Рисунок 11 - Блок-схема используемой функции check\_list

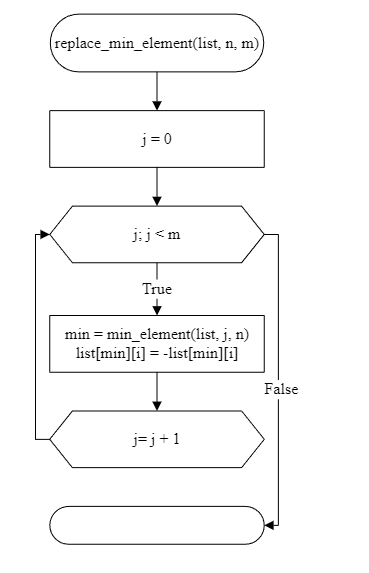
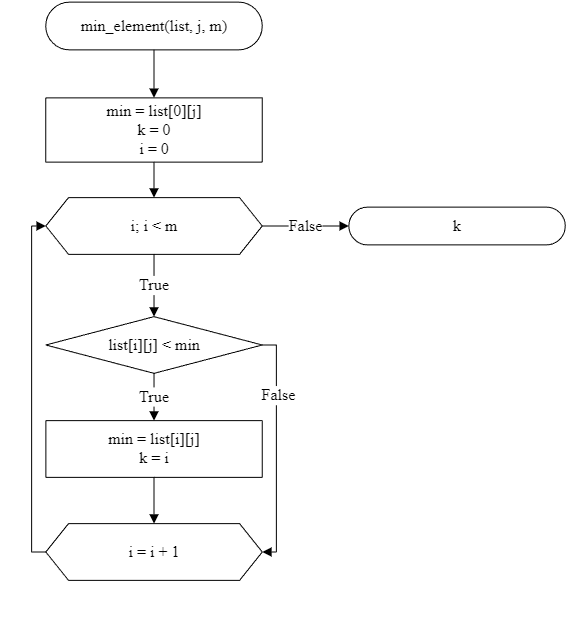


Рисунок 12 - Блок-схема используемой функции replace\_min\_element

Рисунок 13 - Блок-схема используемой функции min\_element

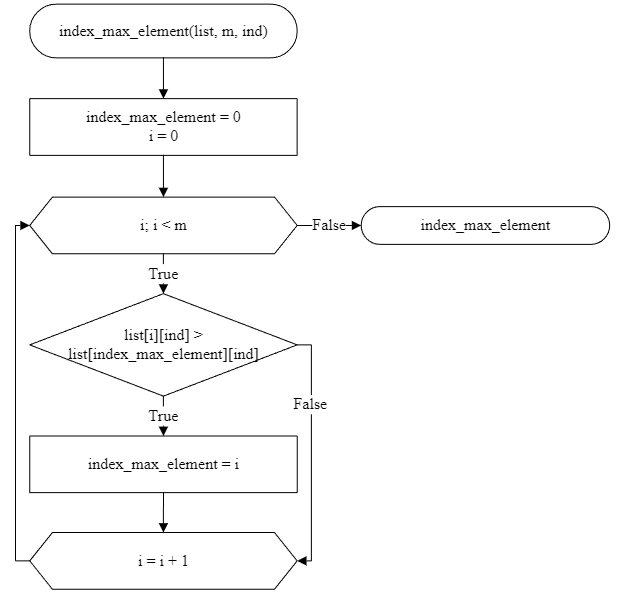


Рисунок 14 - Блок-схема используемой функции index\_max\_element

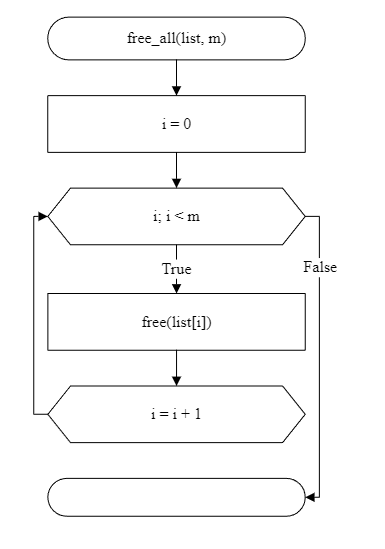


Рисунок 15 - Блок-схема используемой функции free\_all

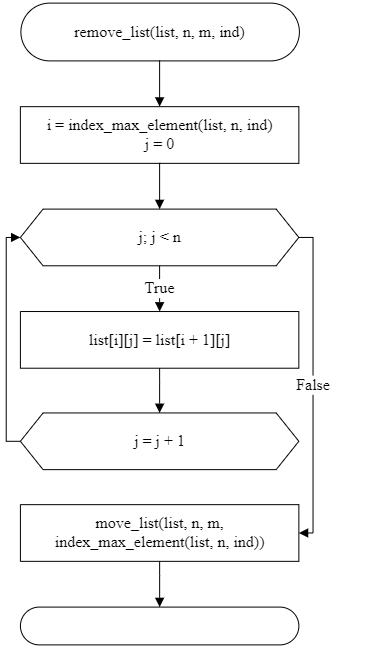
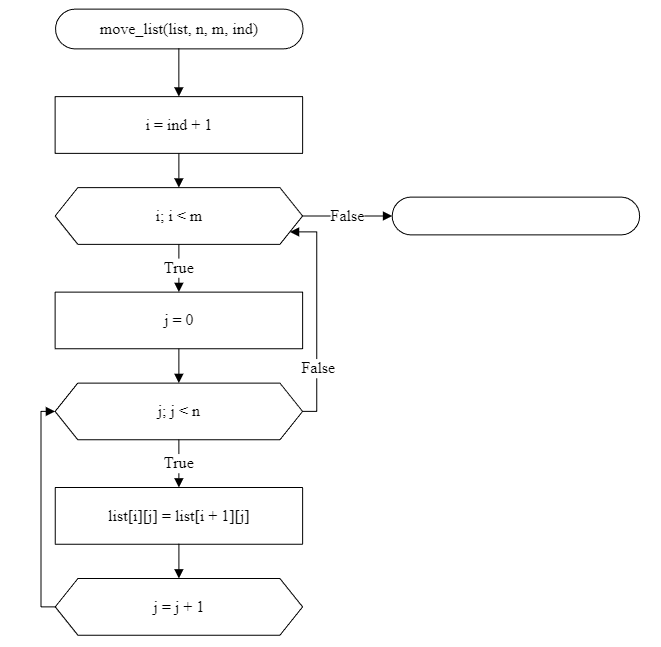
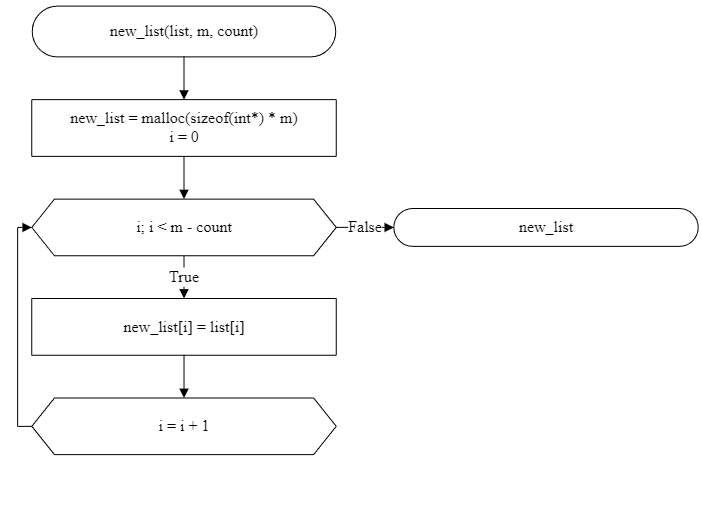


Рисунок 16 - Блок-схема используемой функции remove\_list

Рисунок 17 - Блок-схема используемой функции move\_list

Рисунок 18 - Блок-схема используемой функции new\_list

1. Текст программы на языке C

#include <stdio.h> #include <stdlib.h> #include <stdbool.h> #include <time.h>

/\*\*

@brief input принимает на вход числа типа int, вводимые пользователем

@return возвращает число, которое ввёл пользователь

\*/ int input(void);

/\*\*

@brief initialize\_list создаёт многомерный массив

@param n параметр n

@param m параметр m

@return возвращает указатель на многомерный массив

*/ int*\* initialize\_list(const size\_t n, const size\_t m);

/\*\*

@brief initialize создаёт одномерный массив

@param n параметр n

@return возвращает указатель не одномерный массив

*/ int* initialize(const size\_t n);

/\*\*

@brief manually\_list заполняет двухмерный массив числами, вводимыми пользователем

@param list указатель на двухмерный массив

@param n параметр n

@param m параметр m

*/ void manually\_list(int*\* list, const size\_t n, const size\_t m);

/\*\*

@brief manually\_init заполняет одномерный массив числами, который ввёл пользователь

@param list одномерный массив

@param n

*/ void manually\_init(int* list, const size\_t n);

/\*\*

@brief random\_init заполняет одномерный массив случайными числами

@param list одномерный массив

@param n параметр n

@param min минимальное значение в диапазоне случайных чисел

@param max максимальное значение в диапазоне случайных чисел

*/ void random\_init(int* list, const size\_t n, const int min, const int max);

/\*\*

@brief random\_list заполняет двухмерный массив случайными числами

@param list двухмерный массив

@param n параметр n

@param m параметр m

@param min минимальное значение диапазона случайных чисел

@param max максимальное значение диапазона случайных чисел

*/ void random\_list(int*\* list, const size\_t n, const size\_t m, const int min, const int max);

/\*\*

@brief is\_positive проверяет число на положительных

@param number число

@return вовзращает true, если число положительное и false, если отрицательное

\*/ bool is\_positive(const int number);

/\*\*

@brief input принимает на вход числа типа int, вводимые пользователем

@return возвращает число, которое ввёл пользователь

\*/ int positive\_input(void);

/\*\*

@brief check\_list проверяет, выделилась ли память для массива

@param list двухмерный массив

*/ void check\_list(int*\* list);

/\*\*

@brief check\_list проверяет, выделилась ли память для массива

@param list одномерный массив

*/ void check(int* list);

/\*\*

@brief replace\_min\_element заменяет минимальный элемент в столбце на протовоположный

@param list двухмерный массив

@param n параметр n

@param m параметр m

*/ void replace\_min\_element(int*\* list, const size\_t n, const size\_t m);

/\*\*

@brief min\_element находит индекс минимального элемента в столбце

@param list двухмерный массив

@param m параметр m

@param i параметр i

@return возвращает индекс минимального элемнта в столбце

*/ size\_t min\_element(int*\* list, const size\_t m, const size\_t i);

/\*\*

@brief index\_min\_element находит максимальный элемент в столбце

@param list двухмерный массив

@param n параметр n

@param ind параметр ind

*/ size\_t index\_max\_element(int*\* list, const size\_t n, const size\_t ind);

/\*\*

@brief free\_all освобождает память для всех элементов массива

@param list двухмерный массив

@param m параметр m

*/ void free\_all(int*\* list, const size\_t m);

/\*\*

@brief remove\_list удаляет строчки в массива с максимальными элементами

@param list двухмерный массив

@param n параметр n

@param m параметр m

@param ind параметр ind

*/ void remove\_list(int*\* list, const size\_t n, const size\_t m, const size\_t ind);

/\*\*

@brief move\_list сдвигает строчки массива наверх

@param list двухмерный массив

@param n параметр n

@param m параметр m

@param ind параметр ind

*/ void move\_list(int*\* list, const size\_t n, const size\_t m, const size\_t ind);

/\*\*

@brief new\_list задаёт массиву новое значение с удалёнными строчками

@param list двухмерный массив

@param m параметр m

@param count количество удаленных строчек

*/ int*\* new\_list(int\*\* list, const size\_t m, const int count);

/\*\*

@brief choose перечисляет константы для выбора способа заполнения массива

@param manually ручное заполнение массива

@param random заполнение массива случайными числами

\*/ typedef enum { manually = 1, random }choose;

/\*\*

@brief printArray выводит массив на экран

@param list двухмерный массив

@param n параметр n

@param m параметр m

*/ void printArray(int*\* list, const size\_t n, const size\_t m);

/\*\*

@brief main точка входа в программу

@return возвращает 0 в случае успеха

*/ int main(void)*

*{*

*const size\_t n = positive\_input();*

*const size\_t m = positive\_input();*

*int*\* list = initialize\_list(n, m);

choose init = (choose)input();

switch (init)

{

case manually:

{

manually\_list(list, n, m);

break;

}

case random:

{

const int min = input();

const int max = input();

if (min > max)

{

puts("Error");

return 1;

}

random\_list(list, n, m, min, max);

break;

}

default:

puts("Invalid input");

return 1;

}

replace\_min\_element(list, n, m);  
int count = 0;  
for (size\_t i = 0; i < n; i++)  
{  
 remove\_list(list, n, m, i);  
 count++;  
}  
  
list = new\_list(list, m, count);  
printArray(list, n, m);  
free\_all(list, m);  
free(list);  
return 0;

}

int\*\* new\_list(int\*\* list, const size\_t m, const int count)

{

int\*\* new\_list = (int\*\*)malloc(sizeof(int\*) \* m);

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

{

new\_list[i] = list[i];

}

return new\_list;

}

void move\_list(int\*\* list, const size\_t n, const size\_t m, const size\_t ind)

{

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

{

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

{

list[i][j] = list[i + 1][j];

}

}

}

void replace\_min\_element(int\*\* list, const size\_t n, const size\_t m)

{

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

{

size\_t min = min\_element(list, j, n);

list[min][j] = -list[min][j];

}

}

size\_t min\_element(int\*\* list, const size\_t j, const size\_t m)

{

int min = list[0][j];

size\_t k = 0;

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

{

if (list[i][j] < min)

{ min = list[i][j]; k = i;

}

}

return k;

}

void free\_all(int\*\* list, const size\_t m)

{

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

{

free(list[i]);

}

}

void remove\_list(int\*\* list, const size\_t n, const size\_t m, const size\_t ind)

{

size\_t i = index\_max\_element(list, n, ind);

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

{

list[i][j] = list[i + 1][j];

}

move\_list(list, n, m, index\_max\_element(list, n, ind));

}

size\_t index\_max\_element(int\*\* list, const size\_t m, const size\_t ind)

{

size\_t index\_max\_element = 0;

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

{

if (list[i][ind] > list[index\_max\_element][ind])

{

index\_max\_element = i;

}

}

return index\_max\_element;

}

int input(void)

{

int number = 0;

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

{

puts("Error");

exit(EXIT\_FAILURE);

}

return number;

}

void check\_list(int\*\* list)

{

if (list == NULL)

{

exit(1);

}

}

void check(int\* list)

{

if (list == NULL)

{

exit(1);

}

}

int\*\* initialize\_list(const size\_t n, const size\_t m)

{

int\*\* list = (int\*\*)malloc(sizeof(int\*) \* m);

check\_list(list);

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

{

list[i] = initialize(n);

}

return list;

}

int\* initialize(const size\_t n)

{

int\* list = (int\*)malloc(sizeof(int) \* n); check(list);

return list;

}

void manually\_init(int\* list, const size\_t n)

{

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

{

list[i] = input();

}

}

void manually\_list(int\*\* list, const size\_t n, const size\_t m)

{

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

{

manually\_init(list[i], n);

}

}

void random\_init(int\* list, const size\_t n, const int min, const int max)

{

srand((unsigned int)time((time\_t\*)NULL));

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

{

list[i] = (rand() % max - min + 1) + min;

}

}

void random\_list(int\*\* list, const size\_t n, const size\_t m, const int min, const int max)

{

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

{

random\_init(list[i], n, min, max);

}

}

bool is\_positive(const int number)

{

return number > 0;

}

int positive\_input(void)

{

int number = input();

if (!is\_positive(number))

{

puts("Your input is uncorrected");

exit(1); }

return number;

}

void printArray(int\*\* list, const size\_t n, const size\_t m)

{

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

{

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

{

printf("list[%zu][%zu]= %d\n", i, j, list[i][j]);

}

}

}

1. Результаты выполнения программы

Результаты выполнения программы представлены ниже (Рисунок 19, Рисунок 20, Рисунок 21, Рисунок 22, Рисунок 23).

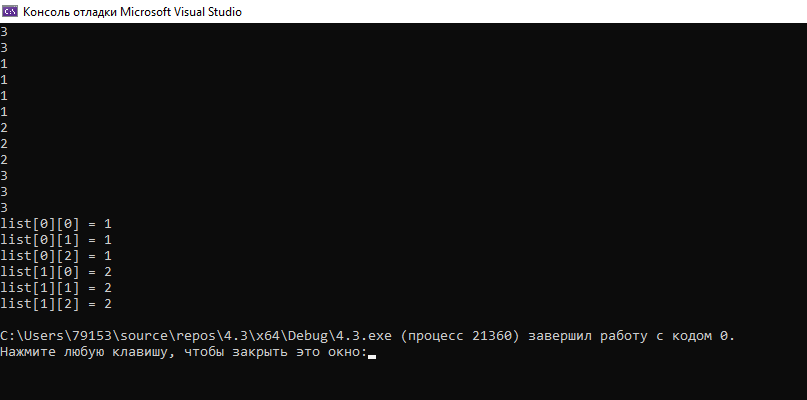


Рисунок 19 - Результаты выполнения программы

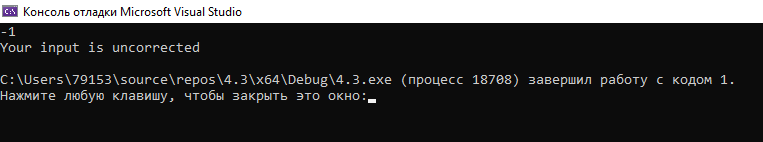


Рисунок 20 - Ошибка, выведенная программа при неверном вводе значений

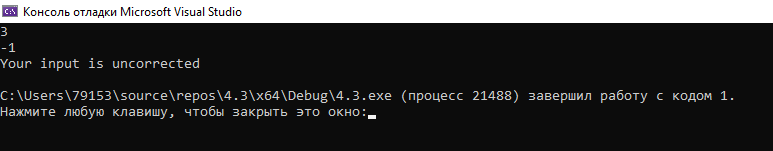


Рисунок 21 - Ошибка, выведенная программа при неверном вводе значений

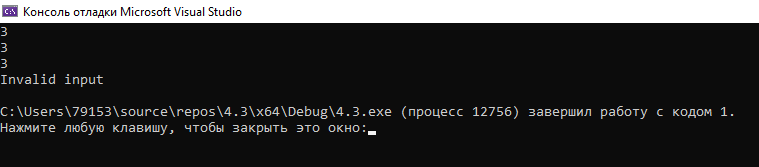


Рисунок 22 - Ошибка, выведенная программа при неверном вводе значений

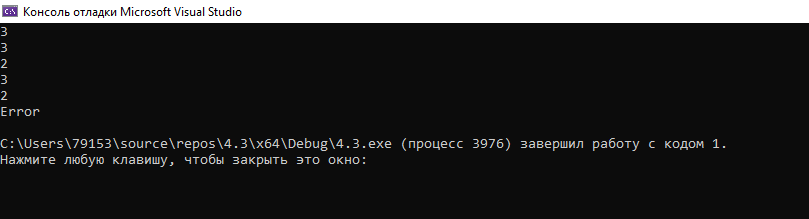


Рисунок 23 - Ошибка, выведенная программа при неверном вводе значений

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