## Балтийский государственный технический университет «ВОЕНМЕХ» им. Д. Ф. Устинова

Кафедра О7 «Информационные системы и программная инженерия»

**Практическая работа №5**по дисциплине «Информатика: Основы программирования»  
на тему «Функции»  
  
Вариант 5

Выполнил:  
Студент Вяткин Н.А.  
Группа О722Б  
  
Преподаватель:  
Назарова М.А.

Санкт-Петербург  
2022 г.

1. Определить, сколько в матрице А (8х6) строк, среднее арифметическое значение элементов которых больше среднего арифметического значения элементов массива В (18).

*Исходные данные:* Матрица а размера 8х6,тип элементов int, способ выделения памяти статический; Массив b из 18 элементов, тип элементов int, способ выделения памяти статический;

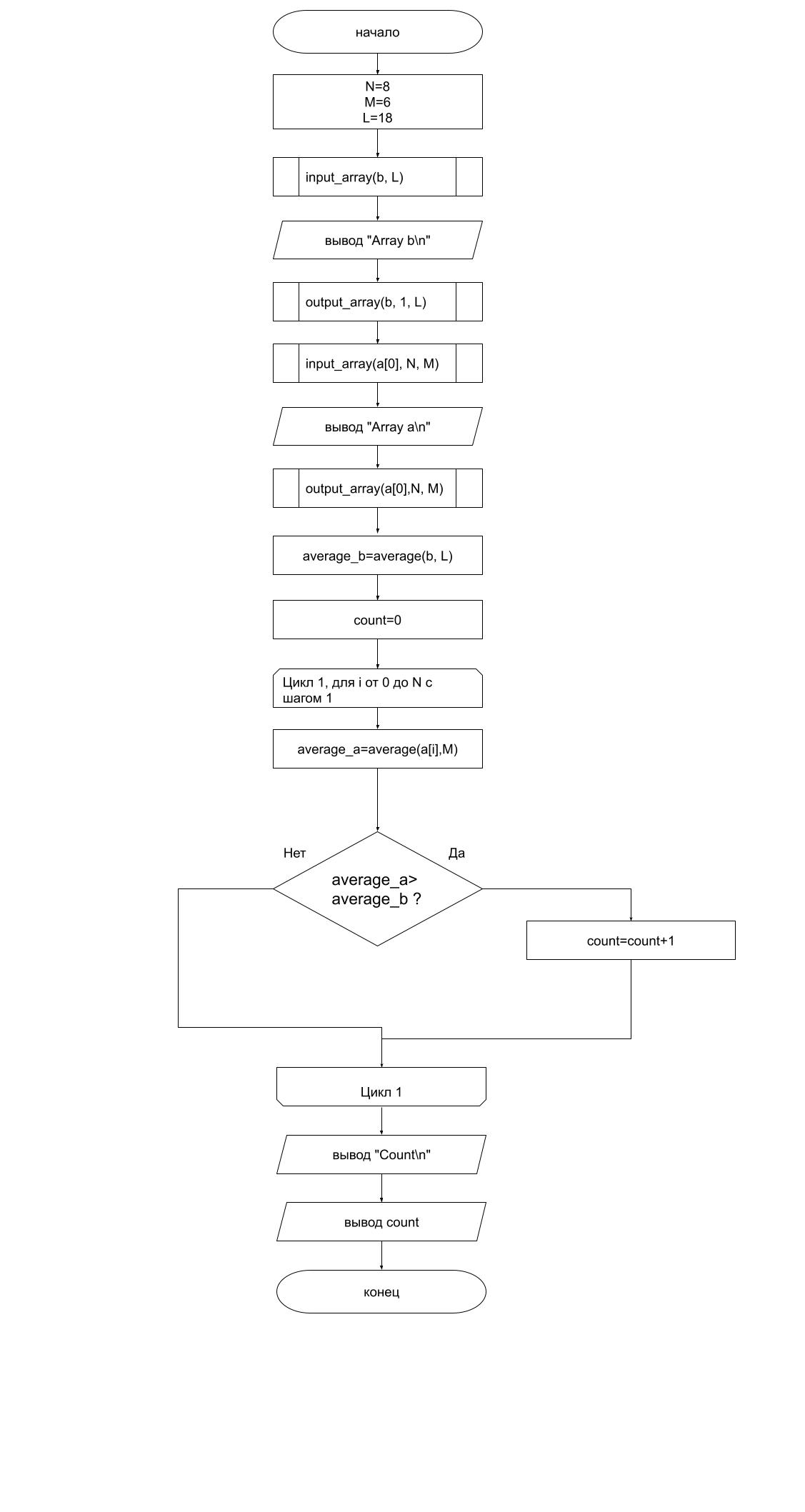
*Результирующие данные:* переменная count тип int, количество строк матрицы а, в которых среднее арифметическое больше среднего арифметического массива b;

*Вспомогательные переменные:* переменнаяaverage\_a, запоминает среднее арифметическое строки матрицы а для сравнения и вывода на экран, тип double; average\_b, запоминает среднее арифметическое массива b для сравнения, тип double; i индекс для цикла сравнения среднего арифметического строк матрицы a с средним арифметическим массива b, тип int;

Таблица тестирования:

|  |  |  |
| --- | --- | --- |
| **Входные данные** | **Ожидаемый результат** | **Результат работы программы** |
|  | 5 |  |
|  | 2 |  |
|  | 6 |  |

Основной алгоритм:



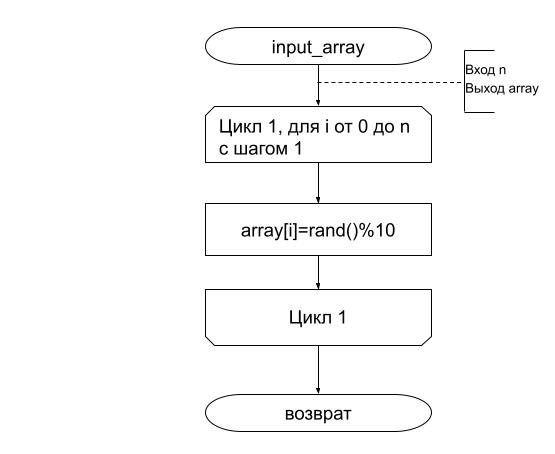
Вспомогательные алгоритмы:

1. Алгоритм ввода массива input\_array

*Входные данные:* имя массива, размер.

*Результирующие данные:* заполненный массив

Схема алгоритма:

**

Этот алгоритм описывается в программе функцией

void input\_array(int \*, int)

*Параметры:*

первый параметр *–* адрес первого элемента массива

второй параметр *–* количество элементов массива

*Возвращаемое значение –* отсутствует

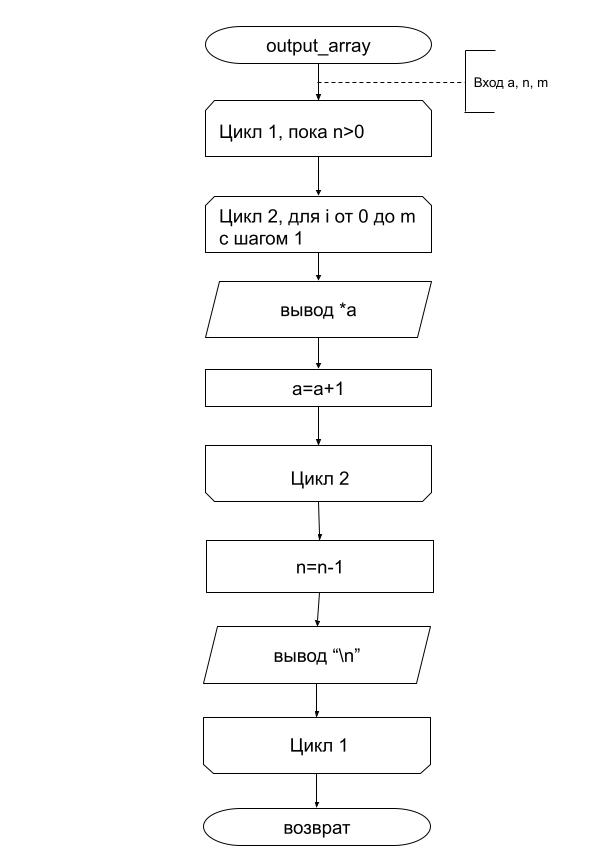
*Вспомогательные переменные:* индекс i, тип int, необходим для цикла заполнения массива элементами;

1. Алгоритм вывода массива

*Входные данные:* имя массива, размер.

*Результирующие данные:* нет

Схема алгоритма:



Этот алгоритм описывается в программе функцией output\_array

void output\_array(const int\*, int, int)

*Параметры:*

первый параметр *–* адрес первого элемента массива

второй параметр *–* количество строк массива

третий параметр *–* количество столбцов массива

*Возвращаемое значение* отсутствует.

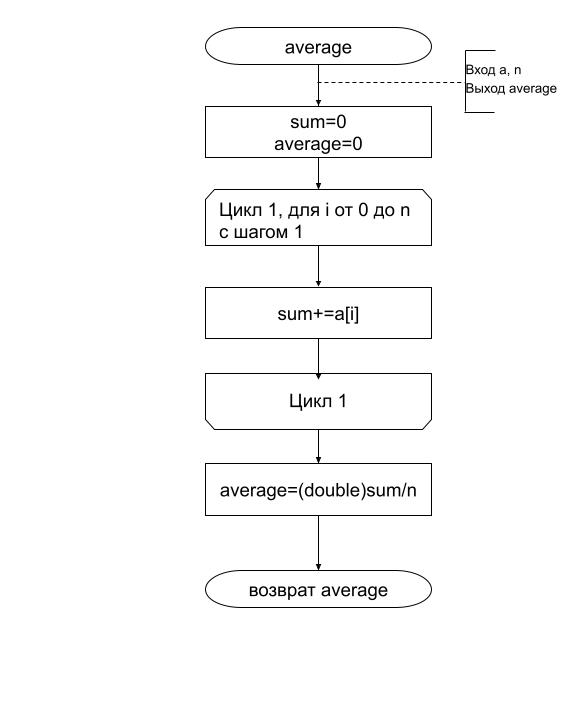
*Вспомогательные переменные:* индекс i, тип int, необходим для цикла вывода массива;

1. Алгоритм вычисления среднего арифметического в строке

*Входные данные:* имя массива, размер.

*Результирующие данные:* average

Схема алгоритма:



Этот алгоритм описывается в программе функцией average

double average(int \*, int)

*Параметры:*

первый параметр – адрес первого элемента массива или первого элемента строки матрицы;

второй параметр – количество столбцов (количество элементов в строке)

*Возвращаемое значение –* average-среднее арифметическое строки, тип double

*Вспомогательные переменные:* индекс i, тип int необходим для цикла, sum тип int переменная необходима для подсчета суммы всех элементов одномерного массива или строки матрицы (тип int, так как все элементы в массиве и матрице типа int), переменная average, тип double, так как среднее арифметическое.

Текст программы

#include <stdio.h>

#include <stdlib.h>

#define N 8

#define M 6

#define L 18

double average(int \*, int);//подсчет среднего арифметического

void input\_array(int \*, int);//заполнение массива

void output\_array(const int \*, int, int);//вывод массива

int main()

{

int a[N][M], b[L];

double average\_a, average\_b;

int i, count;//count переменная, которая считает в скольки строчках матрицы a среднее арифметическое больше среднего арифметического массива b

input\_array(b, L);//заполняем одномерный массив b случайнми числами

printf("Array b\n");

output\_array(b,1,L);//выводим массив b на экран, 1 так одномерный массив из одной строки

input\_array(a[0], M\*N);//заполняем случайными числами двумерный массив а, M\*N количество элементов в массиве а

printf("Array a\n");

output\_array(a[0], N, M);//выводим двумерный массив а на экран

average\_b=average(b, L);//присваиваем переменной average\_b среднее арифметическое массива b

count=0;//обнуляем количество

for (i=0;i<N;i++)/\*получаем среднее арифметическое кажой строки матрицы а и считаем количество строк

, где среднее арифметическо строки больше среднего арифметического массива b\*/

{

average\_a=average(a[i],M);//присваиваем переменной average\_а среднее арифметическое строки с индексом i

if (average\_a>average\_b)//сравниваем среднее арифметическое строки с среднем арифметическим массива b

count+=1;//если больше, то увеличиваем колтичество

}

printf("Count\n");

printf("%3d", count);

return 0;

}

//заполняем массив случайными числами

void input\_array(int \*array, int n)

{

int i;

for (i=0;i<n; i++)

array[i]=rand()%10;

}

//выводим массив на экран с помощью указателя

void output\_array(const int \*a, int n, int m)

{

int i;

for (;n>0;n--)//если количество строк больше нуля, то выводим значения этой строки

{

for(i=0;i<m;i++)

printf("%3d ", \*a++);

printf("\n");

}

}

//подсчет среднего арифмитического в строке

double average(int \*a, int n)

{

int i, sum=0; //обнуляем переменные суммы строки

double average=0;//обнуляем переменную среднего арифметического

for (i=0; i<n; i++)

sum+=a[i];//считаем сумму всех элементов в строке, n-количество элементов в строке

average=(double)sum/n;//вычисляем среднее арифметическое, явно приводим sum к double

return average;

}

1. Удалить из матрицы А (5х7) все столбцы, содержащие не менее двух отрицательных элементов, а из матрицы В (7х7) – содержащие не менее трех отрицательных элементов, передвинув на их место следующие столбцы без нарушения порядка их следования.

*Исходные данные:* Матрица А размера 5х7(N x M) ,тип элементов int, способ выделения памяти динамическая матрицы; Матрица B размера 7х7(L x L) ,тип элементов int, способ выделения памяти динамическая матрицы;

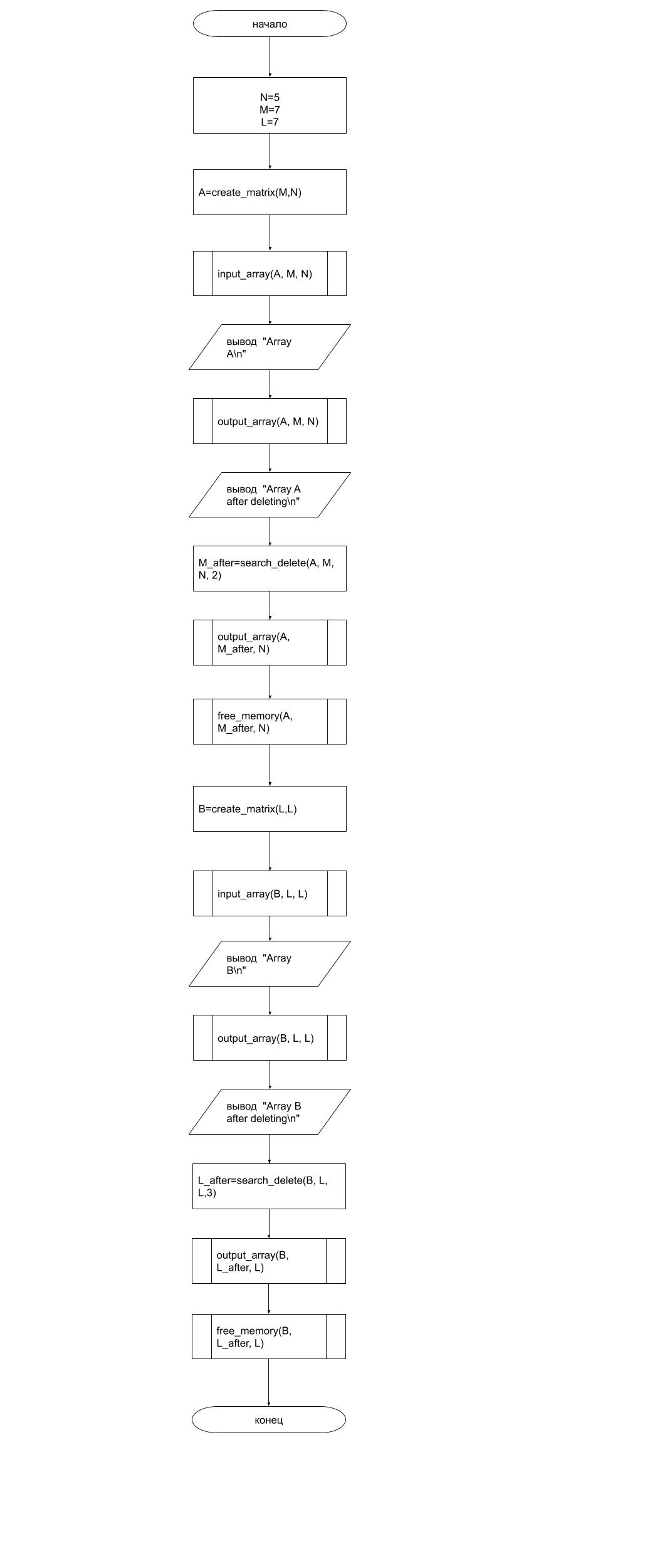
*Результирующие данные:* Измененная матрица А и измененная матрица B, без столбцов в которых количество отрицательных чисел недопустимо.

*Вспомогательные переменные:* M\_after переменная необходима для того чтобы сохранить количество оставшихся столбцов в матрице А после удаления столбцов неудовлетворяющих условию, тип int. L\_after переменная необходима для того чтобы сохранить количество оставшихся столбцов в матрице B после удаления столбцов неудовлетворяющих условию, тип int.

Таблица тестирования:

|  |  |  |
| --- | --- | --- |
| **Входные данные** | **Ожидаемый результат** | **Результат работы программы** |
|  | 5 2 -2 7 6  0 2 3 3 5  -1 3 0 5 -1  0 1 0 0 4  3 5 4 -1 7  0 5 7 3 2 1 -1  0 1 1 2 -1 -1 1  6 5 2 0 5 5 7  1 -1 7 6 4 3 -2  0 6 4 -2 0 2 6  4 3 -2 7 -2 -2 4  -1 1 6 7 1 2 2 |  |
|  | 17  12  41  -48  45  Матрица пустая |  |

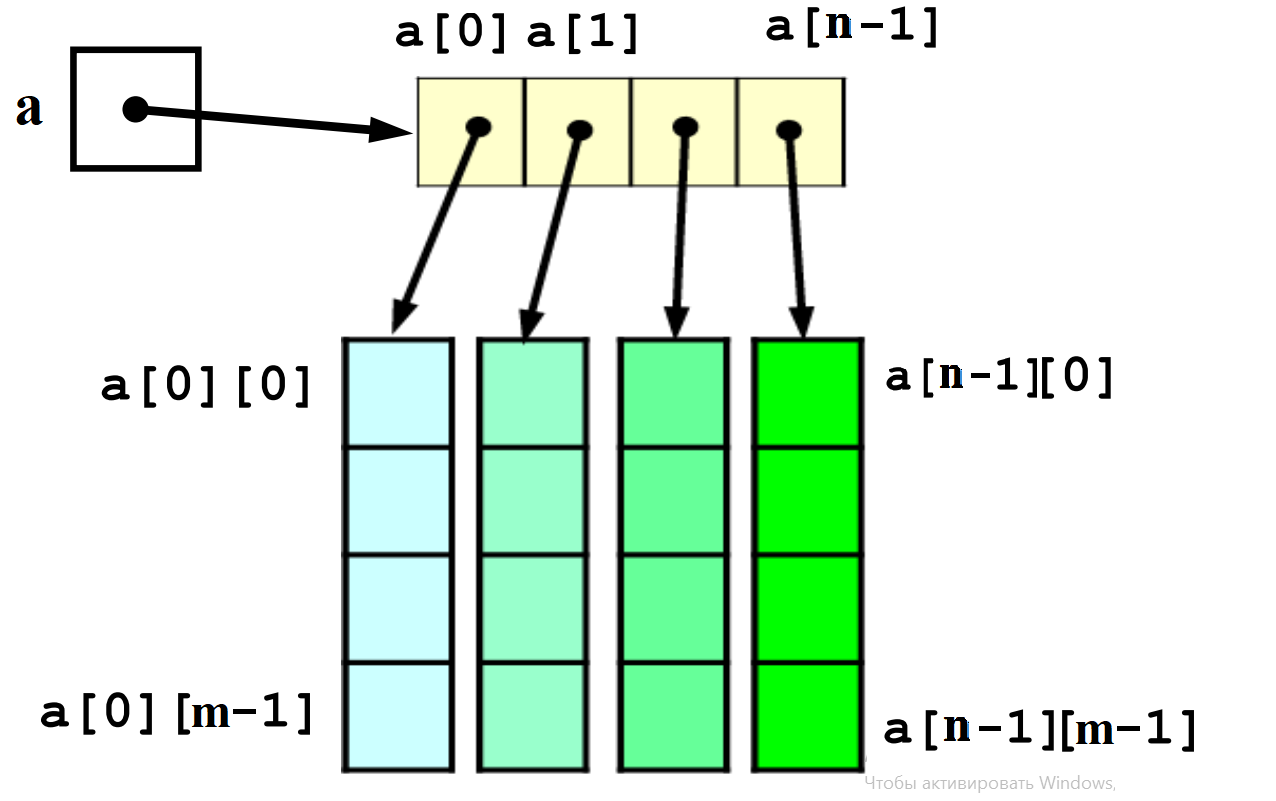
Основной алгоритм:



Вспомогательные алгоритмы:

1. Алгоритм выделения памяти под динамическую матрицу по столбцам

Схема:



Этот алгоритм описывается в программе функцией create\_matrix

int \*\*create\_matrix(int, int);

*Параметры:*

первый параметр – количество строк

второй параметр – количество столбцов

Возвращаемое значение – адрес первого элемента в массиве указателей или NULL в случае отсутствия искомого значения.

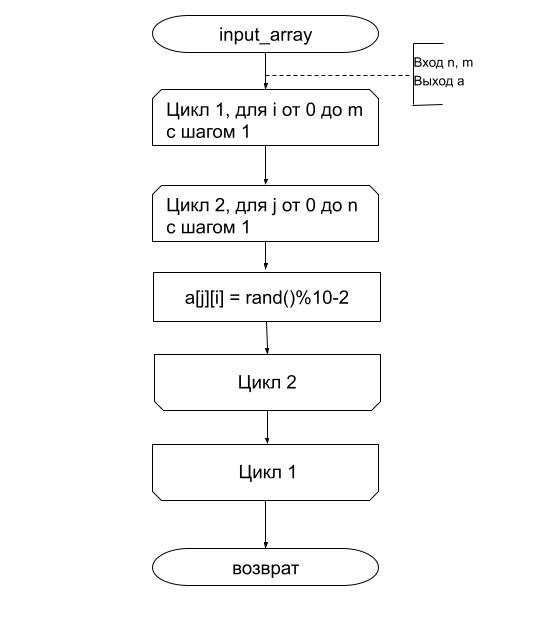
Вспомогательные переменные: динамическая матрица a типа \*\*int, необходима для выделения памяти под динамическую матрицу, индекс i типа int необходим для циклов выделения памяти.

1. Алгоритм ввода массива по столбцам

*Входные данные*: имя массива, размер (количество строк и столбцов);

*Результирующие* данные: заполненный массив

Схема алгоритма:



Этот алгоритм описывается в программе функцией input\_array

void input\_array(int \*\*, int, int);

*Параметры:*

первый параметр –адрес первого элемента динамической матрицы

второй параметр – количество столбцов в матрице

третий параметр – количество строк в матрице

Возвращаемое значение – отсутствует

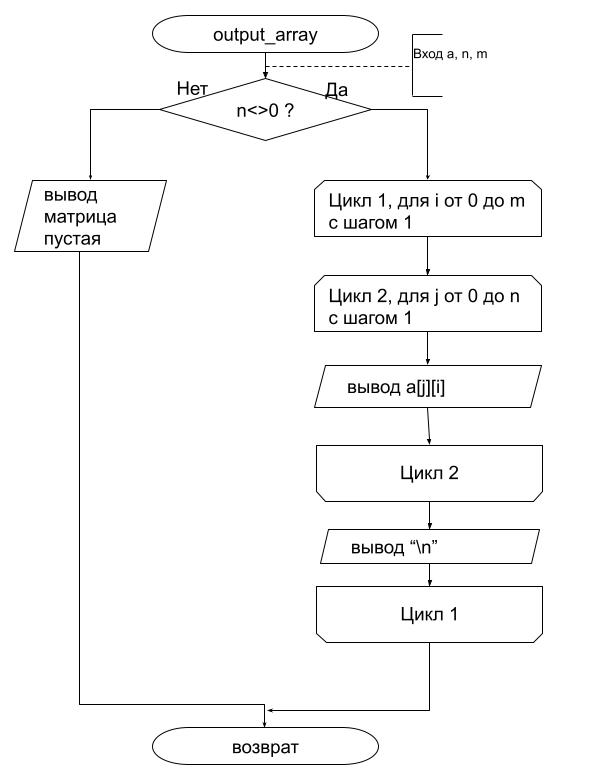
Вспомогательные переменные: i, j индексы типа int, необходимы для циклов заполнения динамической матрицы.

1. Алгоритм вывода массива по столбцам

*Входные данные*: имя массива, размер (количество строк и столбцов);

*Результирующие* данные: нет;

Схема алгоритма:



Этот алгоритм описывается в программе функцией output\_array

void output\_array(const int\*\*, int, int);

*Параметры:*

первый параметр –адрес первого элемента динамической матрицы

второй параметр – количество столбцов в матрице

третий параметр – количество строк в матрице

Возвращаемое значение – отсутствует

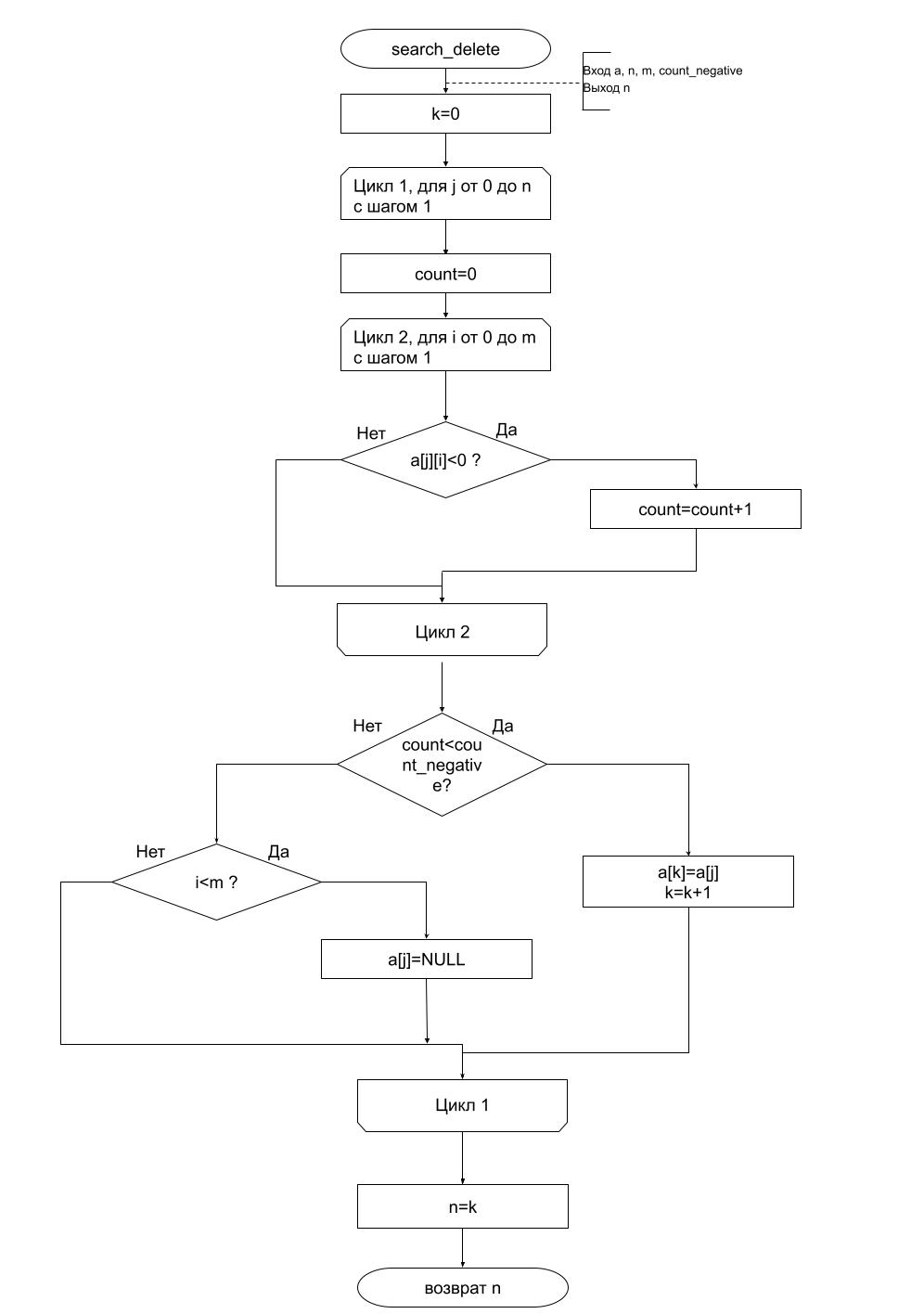
Вспомогательные переменные: i, j индексы типа int, необходимы для циклов вывода динамической матрицы.

1. Алгоритм поиска столбцов с недопустимым количеством отрицательных чисел и удаления этих столбцов

*Входные данные:* имя массива, размер (количество строк и столбцов), недопустимое количество отрицательных чисед в столбце.

*Результирующие данные:* n-количество оставшихся столбцов в массиве после удаления, тип int

Схема алгоритма:



*Параметры:*

первый параметр –адрес первого элемента динамической матрицы

второй параметр – количество столбцов в матрице

третий параметр – количество строк в матрице

четвертый параметр- недопустимое количество отрицательных чисел в столбце

Возвращаемое значение – n-количество оставшихся столбцов в массиве после удаления.

Вспомогательные переменные: i, j индексы типа int, необходимы для циклов просмотра и обработки матрицы, count-подсчитывает количество отрицательных чисел в столбце, тип int, k подсчитывает количество оставшихся столбцов, тип int;

1. Алгоритм освобождения памяти из под динамической матрицы

*Входные данные:* имя массива, количество столбцов оставшиеся в матрице после удаления

*Результирующие данные:* отсутствует

Схема алгоритма:

*Параметры:*

первый параметр –адрес первого элемента динамической матрицы

второй параметр – количество столбцов оставшиеся в матрице в матрице

Возвращаемое значение – отсутствует

Вспомогательные переменные: i индекс для цикла освобождения памяти

Текст программы

#include <stdio.h>

#include <stdlib.h>

int \*\*create\_matrix(int, int);//выделение памяти под динамическую матрицу по столбцам

void input\_array(int \*\*, int, int);//заполнение матрицы случайными числами по столбцам

void output\_array(const int\*\*,int,int);// вывоод матрицы по столбцам

int search\_delete(int \*\*, int, int, int);//поиск и удаление столбцов с недопустимым количеством отрицательных чисел, возвращает количество оставшихся столбцов

void free\_memory(int \*\*, int);//освобождает память

int main()

{

int \*\*A, \*\*B;

int N=5, M=7, L=7, L\_after, M\_after;

A=create\_matrix(M,N);//выделяем память динамически под матрицу A

if (A==NULL)

{

printf("Error");

return 100;

}

input\_array(A, M, N);// заполняем матрицу А по столбцам

printf("Array A\n");

output\_array(A, M, N);//выводим матрицу А на экран по столбцам

printf("Array A after deleting\n");

M\_after=search\_delete(A, M, N, 2);/\*запоминаем в переменную M\_after количество столбцов, которое осталось после удаления

столбцов с недопустимым количеством отрицательных чисел count\_negative\_A\*/

output\_array(A,M\_after,N);//выводим все оставшиеся столбцы матрицы А

free\_memory(A,M\_after);// освобождаем память

B=create\_matrix(L,L);//выделяем память динамически под матрицу B

if (B==NULL)

{

printf("Error");

return 100;

}

input\_array(B,L,L);// заполняем матрицу B по столбцам

printf("Array B\n");

output\_array(B,L,L);//выводим матрицу B на экран по столбцам

printf("Array B after deleting\n");

L\_after=search\_delete(B,L,L,3);/\*запоминаем в переменную L\_after количество столбцов, которое осталось после удаления

столбцов с недопустимым количеством отрицательных чисел count\_negative\_B\*/

output\_array(B,L\_after,L);//выводим все оставшиеся столбцы матрицы B

free\_memory(B,L\_after);//освобождаем память

return 0;

}

void input\_array(int \*\*a, int n, int m)//n-столбцы, m-строки,заполнение матрицы случайными числами

{

int i, j;

for (i = 0; i<m; i++)//цикл по строкам

{

for (j = 0; j<n; j++)//цикл по столбца

{

a[j][i] = rand()%10-2;//заполнение по столбцам

}

}

}

int \*\*create\_matrix(int n, int m)//n-столбцы, m-строки, выделение памяти по столбцам для матрицы

{

int \*\*a, i;

a=calloc(n, sizeof(int\*));

if (a==NULL)

{

printf("Error");

return NULL;

}

for (i=0;i<n;i++)

{

a[i]=calloc(m,sizeof(int));

if (a[i]==NULL)

{

for (--i;i>=0;i--)

free(a[i]);

free(a);

printf("Error");

return NULL;

}

}

return a;

}

void output\_array(const int \*\*a, int n, int m)//n-столбцы, m-строки, вывод матрицы на экран по столбцам

{

int i, j;

if (n)//если количество столбцов не 0, то выводим матрицу, иначе выводим матрица пустая

for (i=0;i<m;i++)

{

for (j=0;j<n;j++)

printf("%4d", a[j][i]);//выводим по столбцам

printf("\n");

}

else

printf("Matrix empty\n");

}

int search\_delete(int \*\*a, int n, int m, int count\_negative)//n-столбцы, m-строки, count\_negative-недопустимое количество отрицательных чисел в столбце

{

int i, j, k, count;//k-количество оставшихся столбцов,count-количество отрицательных чисел в столбце

for (j=0, k=0;j<n; j++)//цикл по столбцам

{

count=0;//обнуляем для подсчета отрицательных чисел в столбце

for (i=0;i<m;i++)//цикл по строкам

if (a[j][i]<0)//считаем количество отрицательных чисел в столбце

count++;

if (count<count\_negative)//ищем столбцы, в которых количество отрицательных чисел больше недопустимого

a[k++]=a[j]; //если в столбце количество отрицательных чисел меньше недопустимого количества отрицательных чисел, увеличиваем количество оставшихся столбцов и смещаем его

else//если в столбце количество отрицательных чисел больше недопустимого количества отрицательных чисел, то удаляем этот столбец

if (i<m)

{

free(a[j]);//освобождаем память

a[j]=NULL;//обнуляем указатель

}

}

n=k;//новое число столбцов в матрице

return n;

}

void free\_memory(int \*\*a, int n)//освобождаем память из под динамической матрицы

{

int i;

for (i=0;i<n;i++)

free(a[i]);

free(a);

}

Определить функцию *Integral()* для приближенного вычисления определенного интеграла вида методомпрямоугольников*.* Использовать эту функцию для вычисления значений двух интегралов, передавая подынтегральную функцию в функцию *Integral()* в качестве параметра. *,*N=30 , метод прямоугольников.

Формула прямоугольников

*Исходные данные:*

нижний и верхний пределы интегрирования задаются константами.

подынтегральные функции описываются функциями в программе.

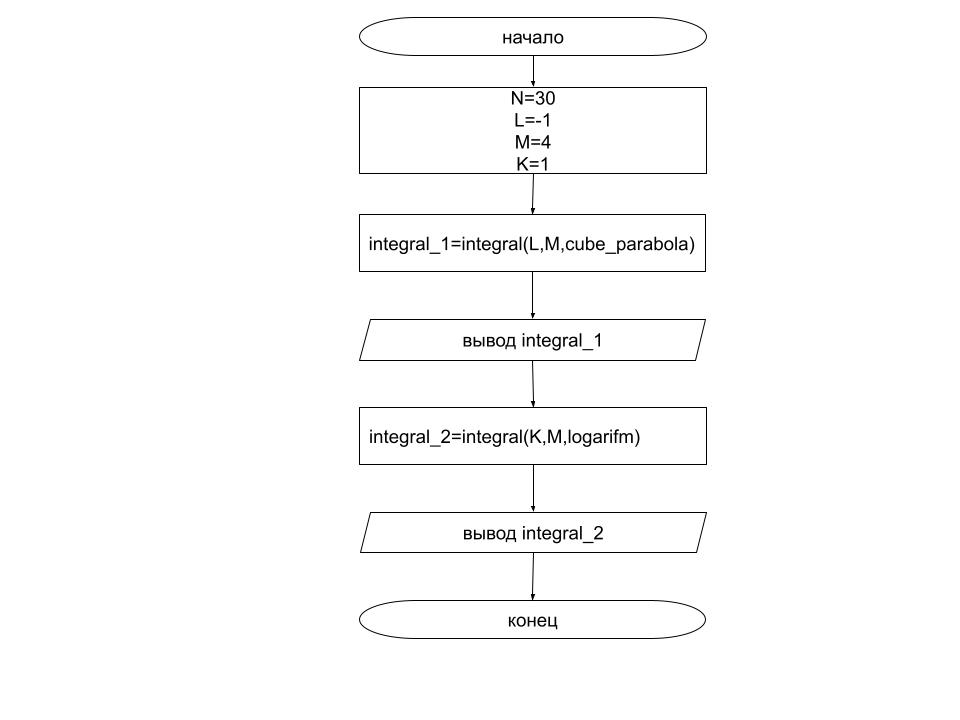
*Результирующие данные:*

значение интеграла – вещественное число

Таблица тестирования:

|  |  |  |
| --- | --- | --- |
| **Входные данные** | **Ожидаемый результат** | **Результат работы программы** |
|  | 142.5 |  |
|  | 5.7 |  |

Схема программы



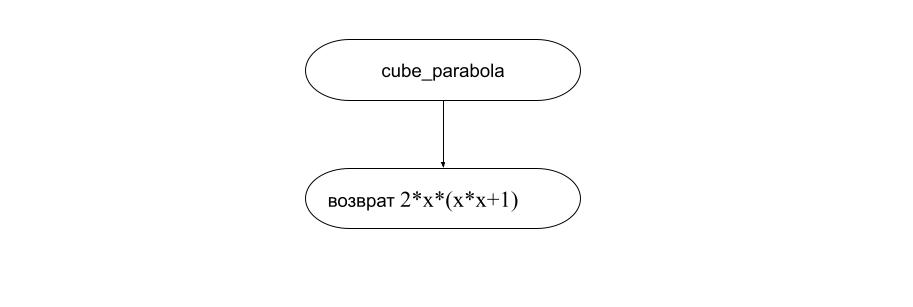
*Вспомогательные алгоритмы:*

1) Алгоритм вычисления

*Входные данные:* x

*Результирующие данные:* значение выражения

Схема алгоритма:



Этот алгоритм описывается в программе функцией

double cube\_parabola(double);

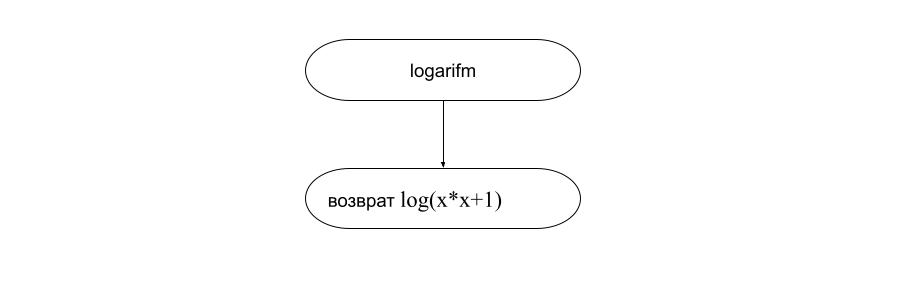
*Возвращаемое значение:* 2\*x\*(x\*x+1)

2) Алгоритм вычисленияln(x2+1)

*Входные данные:* x

*Результирующие данные:* значение выражения

Схема алгоритма:



Этот алгоритм описывается в программе функцией

double logarifm (double);

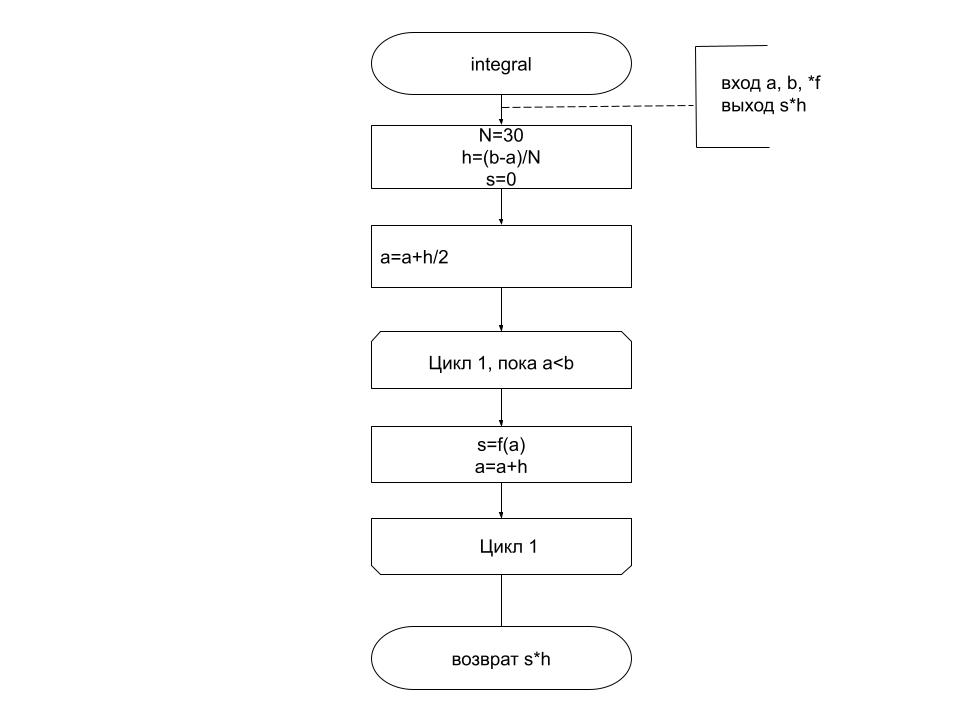
*Возвращаемое значение:* log(x\*x+1)

3) Алгоритм вычисления интеграла методом прямоугольников

*Входные данные:* Нижний предел интеграла и верхний предел интеграла, указатель на функцию подынтегрального выражения

*Результирующие данные:* значение интеграла

Схема алгоритма:



Этот алгоритм описывается в программе функцией

double integral(double, double, double (\*)(double));

*Параметры:*

первый параметр *–* Нижний предел интеграла

второй параметр *–* Верхний предел интеграла

третий параметр *–* Функция вычисления подынтегрального выражения

*Возвращаемое значение* s\*h-значение интеграла;

*Вспомогательные переменные:* h переменная, в которой хранится шаг, тип double, N-количество шагов, тип int, s переменная, которая считает сумму значений подынтегрального выражения на каждом шагу, тип double

Текст программы

#include <stdio.h>

#include <stdlib.h>

#include <math.h>

#define N 30

#define L -1

#define M 4

#define K 1

double cube\_parabola(double);//подсчитывает значение кубической параболы 2\*x^3+1

double logarifm(double);//считает значение ln(x^2+1)

double integral(double, double, double (\*)(double));//вычисляет значение интеграла методом прямоугольников

int main()

{

double integral\_1, integral\_2;

integral\_1=integral(L,M,cube\_parabola);//вычисляем значение интеграла 2\*x^3+1 для x от -1 до 4

printf("%3.5lf\n", integral\_1);

integral\_2=integral(K,M,logarifm);//вычисляем значение интеграла ln(x^2+1) для x от -1 до 4

printf("%3.5lf\n", integral\_2);

return 0;

}

double cube\_parabola(double x)//возвращает значение полученное в результате подстановки x

{

return 2\*x\*(x\*x+1);

}

double logarifm(double x)//возвращает значение полученное в результате подстановки x

{

return log(x\*x+1);

}

double integral(double a, double b, double (\*f)(double))//считает интеграл методом прямоугольников, с помощью 30 точек

{

double h=(b-a)/N, s=0;//h-один шаг,s-считает сумму всех шагов

for (a+=h/2;a<b;a+=h)//цикл подсчета интеграла

s+=f(a);

return s\*h;

}