# Код

#include <iostream>

using namespace std;

void inputArray(int A[4][3]);

void printArray(int A[4][3]);

void printArray(int B[12]);

void transform2DTo1D(int A[4][3], int B[12]);

void count(int B[12], int n, int a, int& equalCount, int& greaterCount);

int main()

{

int A[4][3];

int B[12];

int a;//number to compare elements of array with

int n;//amount of elements of array to compare

int equalCount;

int greaterCount;

inputArray(A);

cout << "Enter intergers n and a\n";

cin >> n >> a;

transform2DTo1D(A, B);

count(B, a, n, equalCount, greaterCount);

printArray(B);

cout << "Number of elements equal to " << a << ": " << equalCount << endl;

cout << "Number of elements greater than " << a << ": " << greaterCount << endl << endl;

printArray(A);

return 0;

}

void inputArray(int A[4][3]) {

cout << "Enter array (4x3) of integers:\n";

for (int i = 0;i < 4;i++) {

for (int j = 0;j < 3;j++) {

cin >> A[i][j];

}

}

}

void printArray(int A[4][3]) {

//prints 2-dimensional array

cout << " A[4][3]:\n";

for (int i = 0;i < 4;i++) {

for (int j = 0;j < 3;j++) {

cout << " " << A[i][j];

}

cout << endl;

}

cout << endl;

}

void printArray(int B[12]) {

//prints 1-dimensional array

cout << " B[12]:\n";

for (int i = 0;i < 12;i++) {

cout << " " << B[i];

}

cout << endl;

}

void transform2DTo1D(int A[4][3], int B[12]) {

//creates 1-dimensional array based on 2-dimensional counterpart

for (int i = 0;i < 4;i++) {

for (int j = 0;j < 3;j++) {

B[4 \* j + i] = A[i][j];

}

}

}

void count(int B[12], int a, int n, int& equalCount, int& greaterCount) {

//count equals

equalCount = 0;

for (int i = 0;i < 12;i++) {

if (B[i] == a) {

equalCount++;

}

}

//count greaters

greaterCount = 0;

for (int i = 0;(i < 2 \* n) && (i < 12);i += 2) {

if (B[i] > a) {

greaterCount++;

}

}

}

# 2.

Проверим работу программы на каком-нибудь входе. Пусть ввод слеующий:

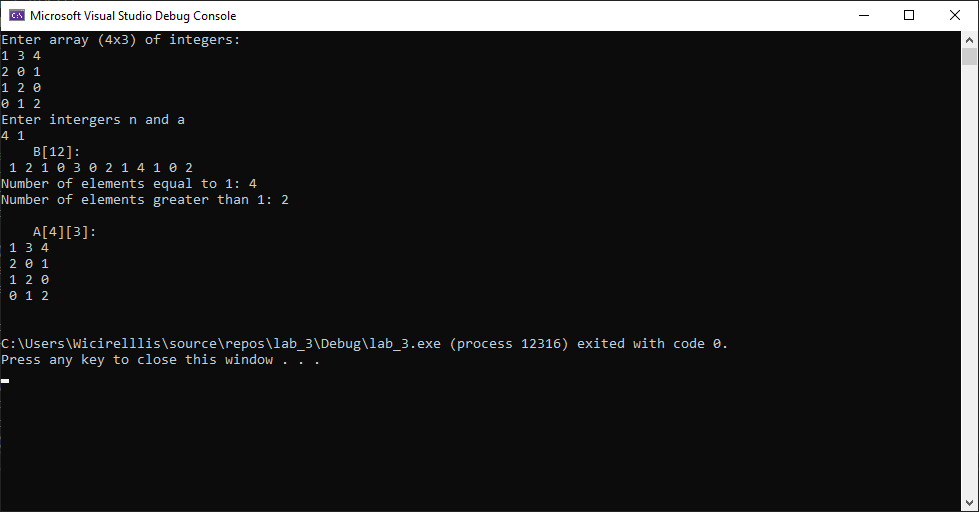
1 3 4

2 0 1

1 2 0

0 1 2

4 1



Number of elements equal to 1: 4

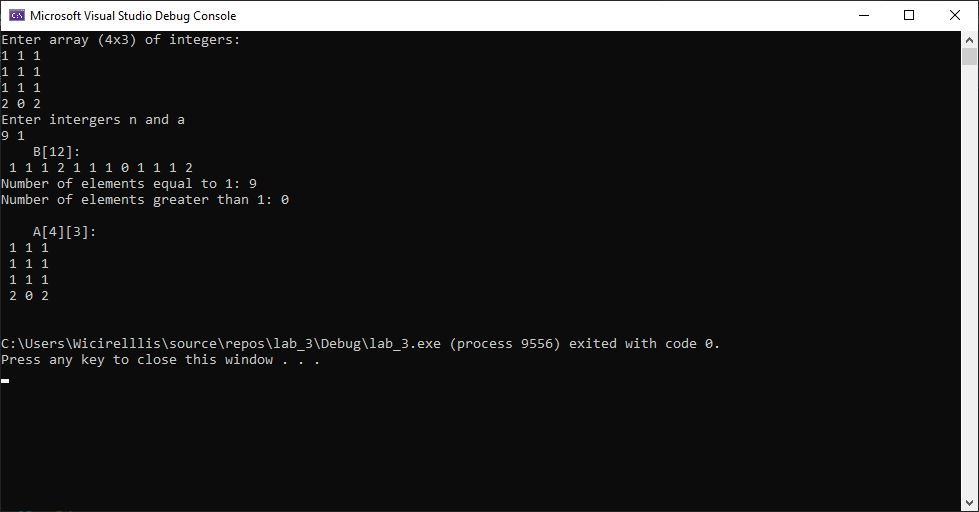
Ровно потому, что в массиве четыре единицы.

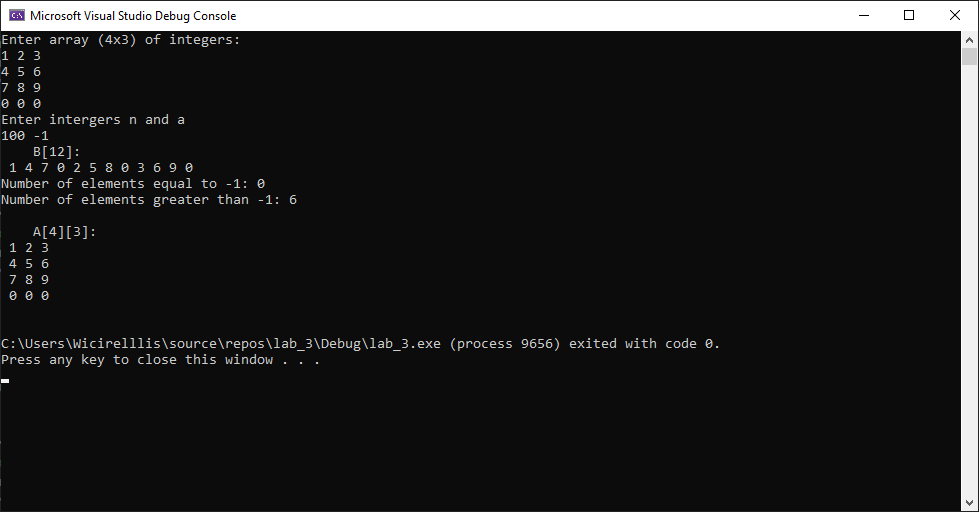
Number of elements greater than 1: 2

Их два, т.к. ищем только среди первых четырех на четных местах. Т.е. среди элементов с индексами 0,2,4,6. Элемент 4 не считается, т.к. его индекс – 8.

# 3.

Несколько скринов вывода





При больших n (n>6) останавливаем поиск на конце массива.

# 4. Вопросы

1) Что такое прототип функции?

Это объявление функции (обычно в саммом начале) без имплементации. Оно сообщает компилятору имя и тип возвращаемого значения функции, а также аргументы и их тип. Это нужно, т.к. сначала компилятору нужно сказать что такая функция есть, и только потом её вызывать.

Пример

int inc(int i);

int n = 10;

inc(n);

int inc(int i) {

return i++;

}

Здесь самая первая строка – это прототип. Нужна ровно потому, что вызов функции идет до реализации (4 строка и 6 соответственно).

2) Что такое указатель? Что хранится в указателе? Как объявляется указатель?

Указатель – это, фактически, адрес переменной. Объявляется через \*. Пример:

int n = 10;

int\* ptr\_n = &n;

Здесь n – это переменная, она хранится в каком-то месте в памяти. И указатель это другая переменная, которая хранит этот адрес.

Стоит помнить, что &n – это взятие адреса, берет переменную и возвращает её адрес. \*ptr\_n – разыменовывание указателя, берет указатель и возвращет переменную, которая по этому адресу лежит.

3) Что выведется в консоль при запуске следующего кода? Подробно опишите, что выводится в консоль.

Скомпилируем код, посмотрим на вывод и поппытаемся понять, почему он такой.

1. #include <iostream>
2. void Fun(int\* k, int\* m)
3. {
4. for (int i = 0; i < 3; i++, m++)
5. {
6. \*m = \*k + 2 \* i \* i;
7. std::cout << m << std::endl << &m << std::endl << \*m << std::endl;
8. }
9. std::cout << k << std::endl << &k << std::endl << \*k << std::endl;
10. }
11. void main()
12. {
13. int M[4]{ 0,0,0,0 };
14. int a = 10;
15. Fun(&a, &M[1]);
16. for (int i = 0; i < 4; i++)
17. {
18. std::cout << M[i] << std::endl << &M[i] << std::endl;
19. }
20. }

В 15 строке объявляется массив из 4 элементов, заполняется нулями.

Затем в 17 строке вызываем fun и передаем в качестве аргументов два указателя (=адреса) – на a и второй элемент массива.

Посмотрим на функцию fun в 3 строке. Она веселая.

Её аргументы – два указателя на инты. Стоит понимать, что k и m – указатели, а \*k и \*m – это инты. Т.е. в 7 строке мы преобразуем не адреса, а интовые переменные.

В 8 строке выводятся m, &m, \*m. Как мы помним, m – это указатель. Поэтому 8 строка печатает указатель (=адрес), указатель на указатель (=адрес указателя или адрес адреса) и инт который лежит по исходному адресу.

Посмотрим что происходит при вызове fun в 17 строке. У нас есть два инта – a = 10 и M[1] = 0. Их адрса передаются в фан, и там меняются сами инты.

Важно, что в 5 строке, в цикле в самом конце i++,m++ Т.е. на каждом шаге цикла у нас на 1 увеличивается не только i, но и m. Но m – это указатель, когда мы прибавляем к нему 1, то получаем адрес рядом. Адрес соседнего элемента массива.

Т.е. при i=0 7 строка это примерно M[1] = a + 2\*0\*0 = 10

При i=1 – M[2] = a + 2\*1\*1 = 12

При i=2 – M[3] = a + 2\*2\*2 = 18

Посмотрим теперь на вывод в консоли. Заметим, что 2,5,8 строки одинаковы. В них печатается &m. Я не знаю, почему они одинаковы. С одной стороны m в цикле инкрементится, m и m+1 указывают на разные части памяти, поэтому можно было бы ожидать что это не одно и то же и лежат по разным адресам. Но, вероятно, дело в том, что тут массив и m+1 это не новый указатель, а исходный указатель m и сдвиг.

Посмотрим на последние восемь строк вывода, это цикл в майне, строка 18. Он просто печатает элементы массива (инты) и их адреса. Кстати видно что они отличаются на 4 (BC+4=C0,C0+4=C4 и т.д), это из-за того, что инт занимает 4 байта в памяти.

