**9.1 2 Dimensional Arrays**

# Declaration of 2D matrices

1. 2D Arrays are declared similar to 1D arrays but with an additional dimension.

Syntax: int arr[rows][columns] For example:

**Int** arr[n][m];

**bool** arr[n][m];

**char** arr[n][m];

**float** arr[n][m];

Code: Taking input or print output of Array of matrx:

#include<bits/stdc++.h>

using namespace std;

int main()

{

    // Taking Input Array

    int n, m;

    cin>>n>>m;

    int arr[n][m];

    for(int i=0; i<n; i++){

        for(int j=0; j<m; j++){

            cin>>arr[i][j];

        }

    }

    // Printing output array

    for(int i=0; i<n; i++){

        for(int j=0; j<m; j++){

            cout<<arr[i][j]<<" ";

        }

        cout<<"\n";

    }

    return 0;

}

**Searching in a matrix**

**Problem**: We have to find if value x is present in the 2D array.

While traversing in the 2D matrix, just we have to put one if statement which checks if(a[i][j] == x) , then x is present otherwise not.

    int x;

    cin>>x;

    bool flag= false;

    for(int i=0; i<n; i++){

        for(int j=0; j<m; j++){

            if(arr[i][j]==x){

                cout<<i<<" "<<j<<" "<<"\n";

                flag=true;

            }

        }

    }

# Spiral Order Matrix Traversal

**Problem**: We have to print the given 2D matrix in the spiral order. Spiral Order means that firstly, first row is printed, then last column is printed, then last row is printed and then first column is printed, then we will come inwards in the similar way.

**Spiral order is given by:**

**1 5 7 9 10** 11 21 41 70 **105 95 81 79 70** 40 15 9 6 **10 12 13** 20 32 **68 63 59** 55 25 29 30 29.

Algorithm: (**We are given 2D matrix of n X m** ).

* 1. We will need 4 variables:

*r\_s -* = **0**. *r\_e -* = **n-1**. *c\_s -* = **0**. *c\_e -* = **m-1**.

* 1. First of all, we will traverse in the row *r\_s* from *c\_s* to *c\_e* and we will increase the *r\_s* with 1 as we have traversed the starting row.
  2. Then we will traverse in the column *c\_e* from *r\_s* to

*r\_e* and decrease the *c\_e* by 1.

* 1. Then we will traverse in the row *r\_e* from *c\_e* to

*c\_s* and decrease the *r\_e* by 1.

* 1. Then we will traverse in the column *c\_s* from *r\_e* to

*r\_s* and increase the *c\_s* by 1.

* 1. We will do the above steps from 2 to 5 until *r\_s* <= *r\_e*

**and** *c\_s* <= *c\_e*.

    int r\_s = 0, r\_e = n-1, c\_s = 0, c\_e = m-1;

    while(r\_s<=r\_e && c\_s<=c\_e){

        // for r\_s

        for(int col=c\_s; col<=c\_e; col++){

            cout<< a[r\_s][col]<<" ";

        }

        r\_s++;

        // for c\_e

        for(int row = r\_s; row <= r\_e; row++){

            cout<<a[row][c\_e]<<" ";

        }

        c\_e--;

        // for r\_e

        for(int col=c\_e; col>=c\_s; col--){

            cout<<a[r\_e][col]<<" ";

        }

        r\_e--;

        // for c\_s

        for(int row=r\_e; row>=r\_s; row--){

            cout<<a[row][c\_s]<<" ";

        }

        c\_s++;

    }

    return 0;

}

**Challenge 1 - Matrix Transpose**

Given a square matrix A & its number of rows (or columns) N, return the transpose of A. The transpose of a matrix is the matrix flipped over it's main diagonal, switching the row and column indices of the matrix.

**Input:**

A = [1,2,3],

    [4,5,6],

    [7,8,9]

**Output:**

[1,4,7],

[2,5,8],

[3,6,9]

#include<iostream>

using namespace std;

int main() {

    int N = 3;

    int A[N][N] = {{1, 2, 3}, {4, 5, 6}, {7, 8, 9}};

    for(int i=0; i<N; i++){

        for(int j=i; j<N; j++){

            //swap

            int temp = A[i][j];

A[i][j] = A[j][i];

            A[j][i] = temp;

        }

    }

   //print transpose for(int i=0; i<N; i++) {

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

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

**Time complexity :** O(N2)

**Space complexity :** O(1), as no extra space for a new matrix was used

**Challenge 2 - Matrix Multiplication**

Given two 2-Dimensional arrays of sizes n1 x n2 and n2 x n3. Your task is to multiply these matrices and output the multiplied matrix.

input:

[2,4,1,2],      [1,2,3]

[8,4,3,6],  X   [4,5,6]

[1,7,9,5]       [7,8,9]

                [4,5,6]

Output:

[33,42,51],

[69,90,111],

[112,134,156]

**Time Complexity: O(n1\*n2\*n3)**

#include<iostream>

using namespace std;

int main()

{   int n1,n2,n3;   // n3 is answer print

    cin >> n1 >> n2 >> n3;

    int m1[n1][n2];

    int m2[n2][n3];

    // input array m1

    for(int i=0; i<n1; i++) {

        for(int j=0; j<n2; j++){

            cin >> m1[i][j];

        }

    }

    // input array m2

    for(int i=0; i<n2; i++) {

        for(int j=0; j<n3; j++){

            cin >> m2[i][j];

        }

    }

    // insalization of answer ans

    int ans[n1][n3];

    for(int i=0; i<n1; i++) {

        for(int j=0; j<n3; j++){

            ans[i][j] = 0;

        }

    }

    for(int i=0; i<n1; i++) {

        for(int j=0; j<n3; j++)

        {

            for(int k=0; k<n2; k++) {

                ans[i][j] += m1[i][k]\*m2[k][j];

            }

        }

    }

    for(int i=0; i<n1; i++) { // print ans

        for(int j=0; j<n3; j++){

            cout << ans[i][j] <<" ";

        }

        cout << endl;

    }

    return 0;

}

**Challenge 3 - 2D matrix Search**

Given n x m matrix. Write an algorithm to find that the given value exists in the matrix or not. Integers in each row are sorted in ascending from left to right. Integers in each column are sorted in ascending from top to bottom

**Solution 1 :** Brute Force Approach **Q. Linear search in a 2D Array**. Time complexity : O(N\*M)

    bool found = false;

    for (int i = 0; i < n; i++) {

        for (int j = 0; j < m; j++) {

            if (mat[i][j] == target){

                found = true;

            }

        }

}

**Solution 2:** Optimised Approach **[IMP]** Time complexity : **O(N + M)**

Start from the top right element.

You are at (r,c) if(matrix[r][c] == target)

return true

If (matrix[r][c] > target)

c--;

else

r++;

At (r,c), you can go to (r-1,c) or (r,c-1), depending on the value of matrix[i][j]

and target.

#include "bits/stdc++.h"

using namespace std;

int main() {

    int n, m;

    cin >> n >> m;

    int target;

    cin >> target;

    int mat[n][m];

    for (int i = 0; i < n; i++) {

        for (int j = 0; j < m; j++) {

            cin >> mat[i][j];

        }

    }

    bool found = false;

    int r = 0, c = n - 1;

    while (r < m && c >= 0) {

        if (mat[r][c] == target) {

            found = true;

        }

        mat[r][c] > target ? c-- : r++;

    }

    if (found){

        cout << "Found";

    }else{

    cout << "Not Found";

    }

}