⇒ 2-D Arrays

→ Representation of 2D array :- → Indexing
↳ Matrix → Grid

	0	1	2	3	4
0	(0,0)	(0,1)	(0,2)	(0,3)	(0,4)
1	(1,0)	(1,1)	(1,2)	(1,3)	(1,4)
2	(2,0)	(2,1)	(2,2)	(2,3)	(2,4)
3	(3,0)	(3,1)	(3,2)	(3,3)	(3,4)

arr

4 x 5 → columns
↳ rows

→ To print
cout << arr [row no.] [column no.];

⇒ declaration

int arr [3][3];

	0	1	2
0	4		
1			8
2			

3 x 3

→ arr [0][0] = 4;
→ arr [1][2] = 8;

Page No.

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

arr

	0	1	2
0	1	2	3
1	4	5	6
2	7	8	9

⇒ int arr[][3] = {12, 34, 56, 78, 91, 23};

↳ Giving column no. is necessary and doesn't matter if you give row no. or not ~~forcefully~~ provided that you are initializing at the time of declaring.

→ Traversal through 2-D arrays

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

	0	1	2
0	1	2	3
1	4	5	6
2	7	8	9

→ To print this array

```
for(int i=0; i<=2; i++){
    for(int j=0; j<=2; j++){
        cout arr[i][j] << " ";
    }
    cout<<endl;
}
```

Date: / /
Page No.

⇒ Taking 2D arrays as input from the user:

- ```
int main(){
    int m;
    cout<<"Enter the no. of rows";
    cin>>m;
    int n;
    cout<<"Enter the no. of columns";
    cin>>n;
    int arr[m][n];

    for(int i=0; i<=m-1; i++){
        for(int j=0; j<=n-1; j++){
            cin>>arr[i][j];
        }
    }
```

⇒ W.A.P to find the largest element of a given 2D array of integers

→ Approach →

```
#include<climits>
int max = INT_MIN;
for(int i=0; i<m; i++){
    for(int j=0; j<n; j++){
        if(max<arr[i][j]){
            max = arr[i][j];
        }
    }
}
cout<<max;
```
```

Date : / /
Page No.

→ WAP to add two matrices

Condition → To add to matrix, the order of both the matrix must be same.

1	2	3
4	5	6

2×3

\+

7	8	9
10	11	12

2×3

=

8	10	12
14	16	18

2×3

a[2][3] b[2][3] res[2]

→
```
int main(){

    int a[2][3] = {1,2,3,4,5,6};
    int b[2][3] = {7,8,9,10,11,12};
    int res[2][3];

    for(int i=0; i<2; i++){
        for(int j=0; j<3; j++){
            res[i][j] = a[i][j] + b[i][j];
        }
    }
}
```

WAP to transpose a matrix

	0	1	2
0	1	2	3
1	4	5	6

2×3

arr[m][n]

⟶

	0	1
0	1	4
1	2	5
2	3	6

3×2

t[n][m]

Date : / /
Page No.

Observation

t[0][0] = arr[0][0]
t[0][1] = arr[1][0]
t[1][0] = arr[0][1]

Q) You are given a matrix / 2D-Array of size (n x n) change this matrix into its transpose

1 2 3
4 5 6
7 8 9

→

1 4 7
2 5 8
3 6 9

	0	1	2	3
0	1	2	3	4
1	5	6	7	8
2	9	10	11	12
4	13	14	15	16

arr[4][4]

→

	0	1	2	3
0	1	↗	↗	↗
1	↙	6	↗	↗
2	↙	↙	11	↗
3	↙	↙	↙	16

swap

arr[4][4]

① While transposing the leading diagonal remains same
(i == j) → copy

	0	1	2	3
0	(0,0)	(0,1)	(0,2)	(0,3)
1	(1,0)	(1,1)	(1,2)	(1,3)
2	(2,0)	(2,1)	(2,2)	(2,3)
3	(3,0)	(3,1)	(3,2)	(3,3)

arr[4][4]

② ~~swap(arr[i][j], arr[j][i])~~

③
```
for (int i=0; i<=n-1; i++){
    for (int j=i+1; j<=n-1; j++){
        swap(arr[i][j], arr[j][i]);
    }
}
```

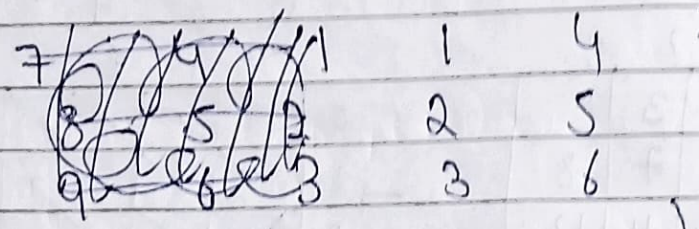
Date : / /
Page No.

1) Rotate a matrix by 90° in clockwise direction. (Leetcode - 48)

→
1	2	3		7	4	1
4	5	6	→	8	5	2
7	8	9		9	6	3

→ Approach

① Take the transpose of the matrix



② Reverse each row

Rotated by 90° ←
7	4	1
8	5	2
9	6	3

→ To reverse

(0,0)	(0,1)	(0,2)
1	4	7
i		j

```
while (i <= j) {
    swap (arr[i], [j])
    i++;
    j--; }
```

Date : / /
Page No.

Q WAP to print the multiplication of two matrices given by the user

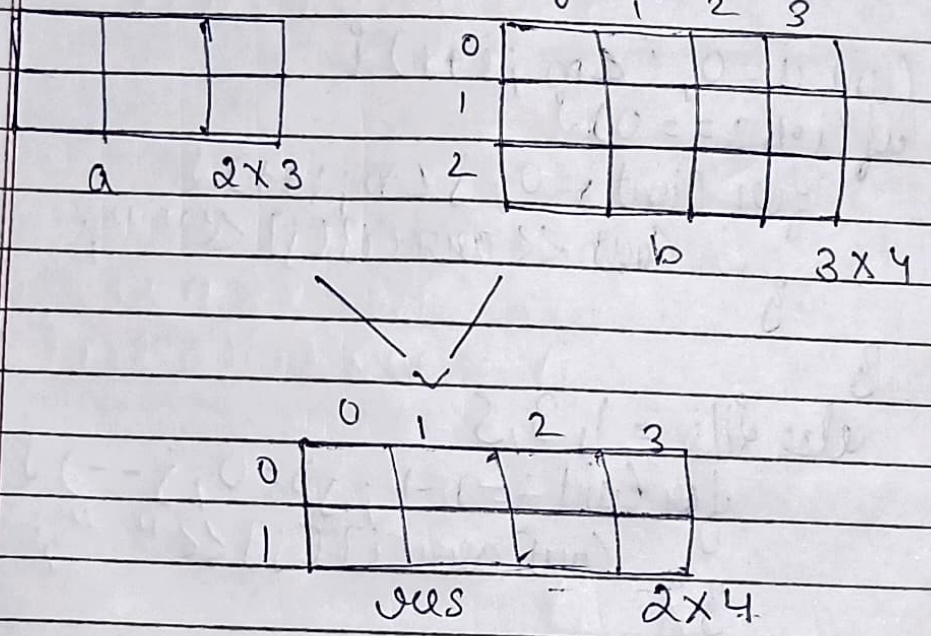
→ Condition for multiplication

arr[m][n] brr[P][Q]

→ [n == P] //

(*) The column of first matrix must be equal to the row of second matrix

→



res[0][0] = a[0][0] * b[0][0] + a[0][1] * b[1][0] + a[0][2] * b[2][0]

res[i][j] = a[i][0] * b[0][j] + a[i][1] * b[1][j] + a[i][2] * b[2][j] //

$$res[i][j] = \sum_{x=0}^{x=n-1} a[i][x] * b[x][j]$$

Date : / /
Page No.

→ WAP to print matrix in wave form

	0	1	2
0	1	2 >	3
1	4	5 <	6
2	7	8 >	9

→ 1 2 3 6 5 4 7 8 9

Approach → even row → normal printing
odd row → reverse printing

- // While printing we have to make different loops.

⇒
```
for (int i=0; i<m; i++) {
    if (i%2==0) {
        for (int j=0; j<n; j++) {
            cout << arr[i][j] << " ";
        }
    }
    else {// i = 1, 3, 5
        for (int j=n-1; j>=0; j--) {
            cout arr[i][j] << " ";
        }
    }
}
```

⇒ Column wise printing —

1	2	3
4	5	6
7	8	9

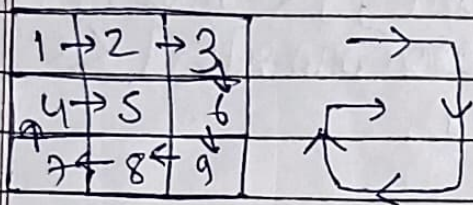
1 4 7 2 5 8 3 6 9

Date : / /
Page No.

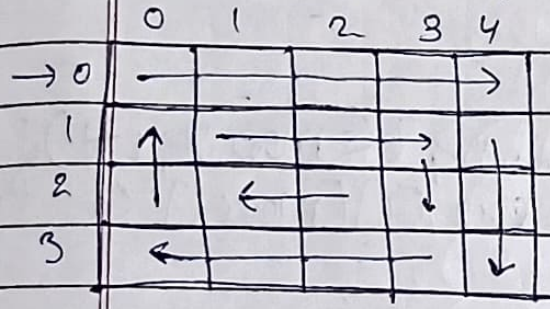
→
```
for(int j=0; j<n; j++){
    for(int i=0; i<m; i++){
        cout<<arr[i][j]<<" ";
    }
}
```

~~Marked~~ : ~~0 0~~

→ WAP to print matrix in spiral (Leetcode 54)



→ 1 2 3 6 9 8 7 4



→ Approach

Min row → minr
Max row → maxr
Min column → minc
Max column → maxc

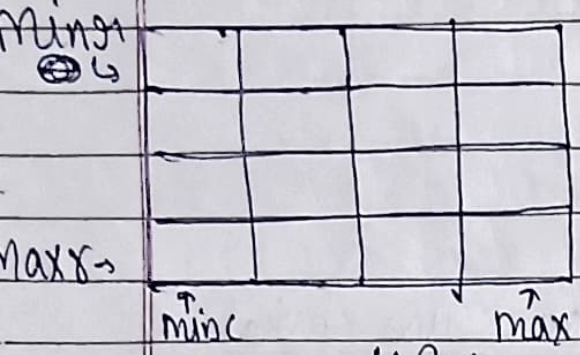
Right → minr fix
minc → maxc
minr ++;

down → maxc fix
minr → maxr;
maxc--;

left → maxr fixed
maxc → minc
maxr--;

Date : / /
Page No.

up → minr, fixed i
maxr → minr;
minr++;




```
// Right
for(int j = minc; j <= maxc; j++){
    cout << arr[minr][j] << " ";
}
minr++;

// Down
for(int i = minr; i <= maxr; i++){
    cout << arr[i][maxc] << " ";
}
maxc--;

// Left
for(int j = maxr; j >= minc; j--){
    cout << arr[maxr][j] << " ";
}
maxr--;

// Up
for(int i = maxr; i >= minr; i--){
    cout << arr[i][minc] << " ";
}
minc++;
```

2-D Vectors

→ Advantages of vectors over arrays

① Increase rows.
② Variable column.

→ V = { {1,2,3}, {4,5}, {6,7,8,9,10} }
0 1 2

	0	1	2	3	4
0	1	2	3		
1	4	5			
2	6	7	8	9	10

→ Passing of 2-D arrays into functions

→ In 2-D arrays while passing it into a function, it is necessary to provide both column & row.

- void change2D (int arr[][]) {
  }
  ↳ ✗

- void change2D (int arr[2][3]) {
  }
  ↳ ✓

→ How to create a 2D vector

→ Next Page

Date: / /
Page No.

```
void main(){
    vector<int> v1;
    v1.push_back(1);
    v1.push_back(2);
    v1.push_back(3);

    vector<int> v2;
    v2.push_back(4);
    v2.push_back(5);

    vector<int> v3;
    v3.push_back(6);
    v3.push_back(7);
    v3.push_back(9);
    v3.push_back(10);

    vector<vector<int>> v;
    v.push_back(v1);
    v.push_back(v2);
    v.push_back(v3);
}
```

→ Declarations of 2D vectors

- `vector<vector<int>> v;`
- `vector<vector<int>> v(m);` ← initial size
- `vector<vector<int>> v(m, vector<int>(n));`
  - `m` → rows
  - `n` → column

Date : / /
Page No.

- `vector<vector<int>> V(3, vector<int>(4, 20));`

  ↓ initial value

→ no. of rows = V.size();

– no. of columns = V[0].size();

Q) Given an integer 'numRows', generate Pascal's triangle.

→ numRow = 5;

	0	1	2	3	4
0	1				
1	1	1			
2	1	2	1		
3	1	3	3	1	
4	1	4	6	4	1

→

	0	1	2	3	4
0	$^0C_0$				
1	$^1C_0$	$^1C_1$			
2	$^2C_0$	$^2C_1$	$^2C_2$		
3	$^3C_0$	$^3C_1$	$^3C_2$	$^3C_3$	
4	$^4C_0$	$^4C_1$	$^4C_2$	$^4C_3$	$^4C_4$

$$\Rightarrow \left[{}^iC_j = \frac{i!}{j! \times (i-j)!} \right] \checkmark$$

→ Or by def^n of Pascal triangle, we know that

$$V[i][j] = V[i-1][j] + V[i-1][j-1];$$

⇒ if (j == 0 || i == j) V[i][j] = 1;
else
V[i][j] = V[i-1][j] + V[i-1][j-1];

Date : / /
Page No.

⇒ Score after flipping matrix (Leetcode (861))

→ Code to convert binary to number

Eg. arr

0	1	2	3	4	5	6
1	1	0	0	1	0	1

6 5 4 3 2 1 0

2^6 2^5 2^4 2^3 2^2 2^1 2^0

1	1	0	0	1	0

```
int sum = 0;
int x = 1;
for (int i=6; i≥0; i--) {
    sum += arr[i] * x;
    x *= 2;
}
cout << sum;
```

(*)

	0	1	2	0	
0	0	0	1	1	→ 0011 → 3
1	1	0	1	0	→ 0010 → 10
2	1	1	0	0	→ 1100 → 12

→ 25

→ We have to make the maximum number but we can either flip a row or column (0's → 1's).

⇒ Approach → (1) Make the leftmost bit 1.

Or

0^{th} column → all ones → rows flip

↓

initially 0

(2) Flip the column where number of 0's are greater than no. of 1's

Date : / /
Page No.

⇒ Search an element in a 2D-Matrix

→ Algorithm

→
```
int i = 0 ;
int j = cols-1;

while (i< rows && j=> 0) {
        if (matrix[i][j] == target) return true;
        else if (matrix[i][j] > target) j--;
        else i++;
}
return false;
```