2 D - Array

creation

1 D => int arr[5]

2 D => row -> 5
col -> 10 int arr[5][10]

2 D => 100 row
1000 col int arr[100][1000]

Initialise

1 D => int arr[] = {10, 20, 30}

2 D => row -> 2
col -> 4 → int arr[2][4] = {{10, 20, 30, 40},
{80, 70, 60, 50},}

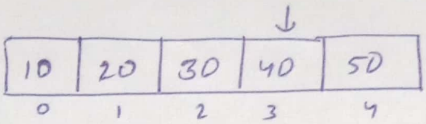
2 D => row -> 3
col -> 5

```
int arr[3][5] = {
    {1, 2, 3, 4, 5},
    {6, 7, 8, 9, 10},
    {10, 20, 30, 40, 50}
}
```

	C_0	C_1	C_2	C_3	C_4
r_0 →	1	2	3	4	5
r_1 →	6	7	8	9	10
r_2 →	10	20	30	40	50

Access

1 D ⇒



arr[index] arr[3]

2 D ⇒

	0	1	2
0	(0,0) 10	(0,1) 20	(0,2) 30
1	(1,0) 40	(1,1) 50	(1,2) 60
2	(2,0) 70	(2,1) 80	(2,2) 90

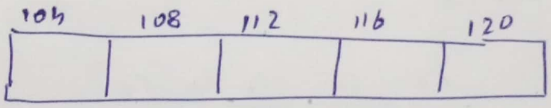
arr[0][0] = 10
arr[0][1] = 20
arr[0][2] = 30
arr[1][0] = 40
arr[1][1] = 50
arr[1][2] = 60
arr[2][1] = 70
arr[2][2] = 80
arr[2][3] = 90

arr[i][j]

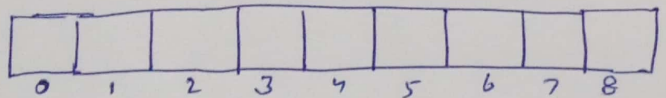
i → row index

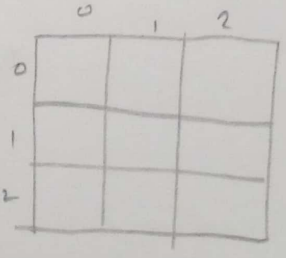
j → col index

int arr[5]



int arr[3][3]





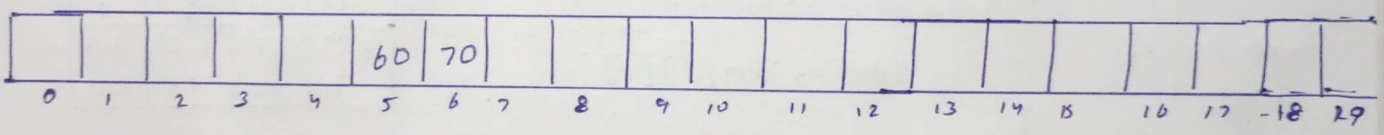
arr [1][1] → formula

$c * i + j$

$3 \times 1 + 1 \Rightarrow (4)$

C → columns

	0	1	2	3
0	10	20	30	40
1	50	60	70	80
2	90	100	110	120
3	130	140	150	160
4	170	180	190	200




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

2D array me col specify krna mandatory h (column size) → because to calculate

$c * i + j$

row wise access

outer

```
for (i=0; i<row; i++){
    for (j=0; j<col; j++){
        cout << arr[i][j]
    }
}
```

	0	1	2	3
0	(0,0) 10	0,1 20	0,2 30	0,3 40
1	1,0 50	1,1 60	1,2 70	1,3 80
2	2,0 90	2,1 100	2,2 101	2,3 110
3	3,0 120	3,1 130	3,2 140	3,3 150

column wise access

outer loop → column

inner loop → row

```
for (i=0; i<col; i++){
    for (j=0; j<row; j++){
        cout << arr[j][i]
    }
}
```

	0	1	2	4
0	(0,0) 1	(0,1) 2	(0,2) 3	(0,4) 4
1	(1,0) 10	(1,1) 20	(1,2) 30	(1,3) 40
2	(2,0) 11	(2,1) 21	(2,2) 13	(2,3) 14
3	(3,0) 20	(3,1) 22	(3,2) 53	(3,3) 47

Searching

	0	1	2
0	10	20	30
1	40	50	60
2	70	80	90

T/F =>

2 LOOP

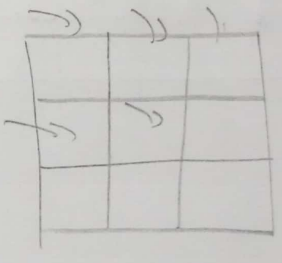
↳ if (arr[i][j] == target)

return true

target => 70

otherwise return false

Max no. in an 2D array



```
int maxAns = INT_MIN;

if (no > maxAns)
    ↳ maxAns = no
```

2 LOOP lagenge

Min no. in an 2D array

```
int minAns = INT_MAX

if (no < minAns)
    ↳ minAns = no
```

Row wise sum

	0	1	2	3	
0	10	20	5	7	→ sum = 42
1	2	4	6	8	→ sum = 20
2	10	15	15	10	→ sum = 50

Pseudo-code

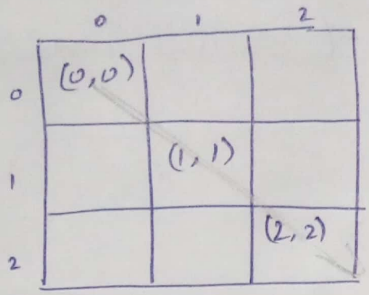
```
foo (i -> i<r){
    int sum=0
    foo (j -> j< c){
        sum += arr[i][j]
    }
    cout<< sum
}
```

column wise sum (H/W)

	0	1	2
0			
1			
2			

```
foo (i -> i<col) {
    int sum=0
    foo (j -> j< row) {
        sum += arr[j][i]
    }
    cout << sum
}
```

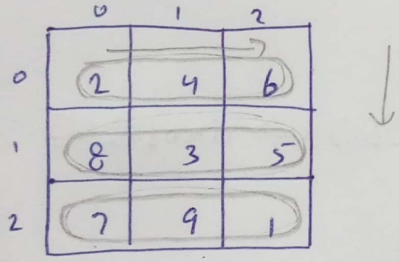
Diagonal sum

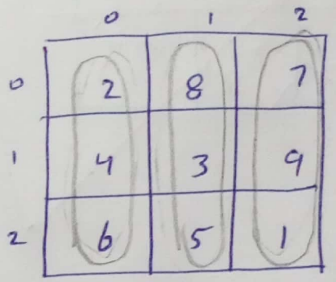


(Assume row == col)

```
sum = 0
for (i=0; i<row ; i++)
    sum += arr[i][j]
}
cout << sum
```

Transpose of a matrix





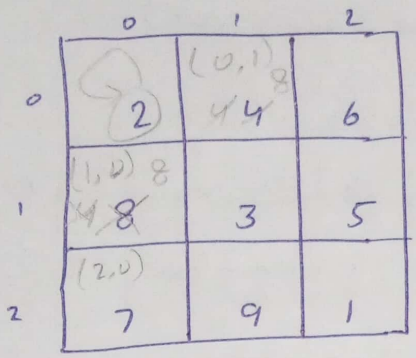
arr[0][0] ⟷ arr[0][0]

arr[1][0] ⟷ arr[0][1]

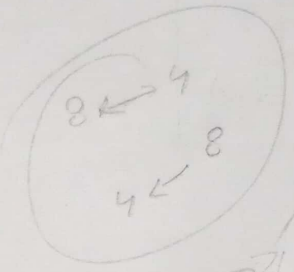
arr[2][0] ⟷ arr[0][2]

arr[2][1] ⟷ arr[1][2]

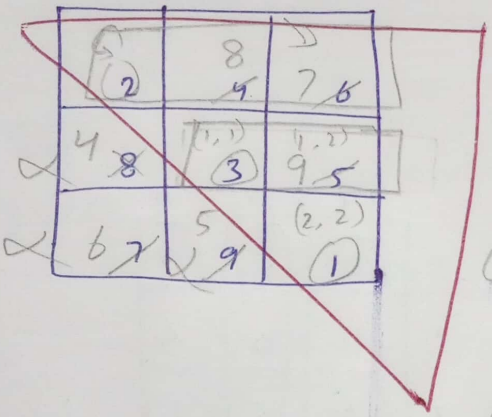
swapping ⟶ arr[i][j] ⟺ arr[j][i]



swap(arr[i][j], arr[j][i])



work only on square matrix



(1) (0, 0)

arr[0][0] ↔ arr[0][0]

(2) (0, 1)

arr[0][1] ↔ arr[1][0]

(3) (0, 2)

arr[0][2] ↔ arr[2][0]

(4) (1, 1)

arr[1][1] ↔ arr[1][1]

(5) (1, 2)

arr[1][2] ↔ arr[1][2]

(6) (2, 2)

arr[2][2] ↔ arr[2][2]

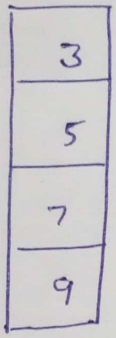
```
for(i=0 ; i<row){
    for(j=i ; j<col)
    }
}
```

Vector

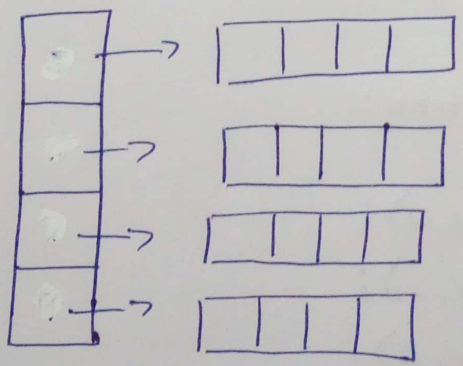
1 D → vector <int> arr

2 D → vector < vector <int>> arr

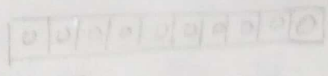
vector <int> v



vector < vector <int>> v

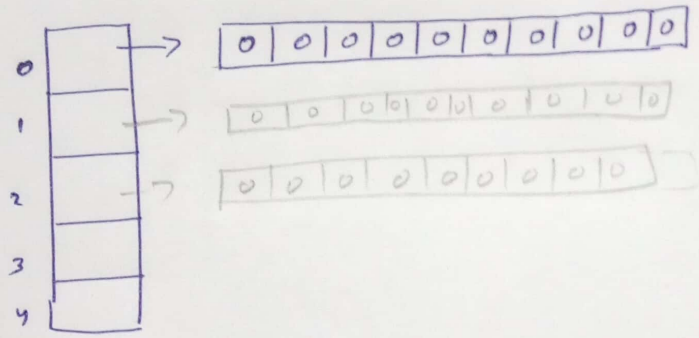


```
vector <vector <int> > arr
```



```
vector <vector <int>>  arr (5, vector <int> (10, 0))
```

- `vector <vector <int>>` → 2D array
- `arr` → name
- `5` → row size
- `vector <int> (10, 0)` → row item
  - ↳ initialise
    - ↳ with a vector of size `10` that
    - ↳ initialise with `0`



H/W ⇒ jagged array

```
vector <vector <int>> boo;

vector <int> vec1 (10, 0);
vector <int> vec2 (5, 1);
vector <int> vec3 (7, 0);
vector <int> vec4 (4, 1);

boo.push_back (vec1);
boo.push_back (vec2);
boo.push_back (vec3);
boo.push_back (vec4);
```