Array Level-3

2-D Arrays

Creation :

1D → int arr[5];

2D → int arr [5][10];
↑ Row ↑ Col

2D → int arr [100] [000]

Initialization
↓
int arr[] = { 10, 20, 30 }
1D ↑

int arr[2][4] = { {10,20,30,40},
{80, 70, 60, 50} }
2D ↑

Access

1D → | 10 | 20 | 30 | 40 | 50 |
0 1 2 3 4

arr[3] = 40

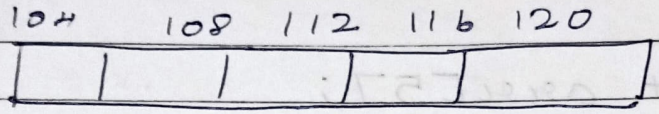
2D →		0	1	2
	0	10	20	30
	1	40	50	60
	2	70	80	90

arr[0][2] = 30 arr[i][j]
↑ row ↑ col

How it is store in Memory?

1-D

int arr[5]



2-D is stored in linear array only we visualize it as matrix.

Formula used ⇒ c*i+j

for initializing 2-D Array, we have to compulsorily give column number because formula contains column no. when we pass array in function it is mandatory to send columnnumber with array.

Row wise Access

code:

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

Columnwise Access

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

Input can be taken in this way only.

Searching in 2D Array

```
bool findTargent(int arr[][4], int row,
        int col, int target) {
    for(int i=0; i<row; i++) {
        for(int j=0; j<col; j++) {
            if(arr[i][j] == target) {
                //if found, return True
                return true;
            }
        }
    }
    //iss line par tabhi aa skte ho,
    //jab saare K saare elements check
    //ho chuke honge and target nahi mila hoga
    //return krdo false
    return false;
}
```

Find maximum & minimum elements in the Array [2D]

Code:

```
int findMAX( int arr[][4], int row,
                    int col) {
  int maxAns = INT_MN;
 for(int i=0; i<row; i++) {
   for(int j=0; j<col; j++) {
    if( arr[i][j] > maxAns) {
      maxAns = arr[i][j];
    }
   }
 }
 return maxAns;
}

int findMin(int arr[][4], int row,
 int col). {
  int minAns = INT_MXX;
 for(int i=0; i<row ;i++) {
      for(int j=0 ;j<col; ;++) {
    if(arr[i][j] < minAns ) {
      minAns = arr[i][j];
    }
   }
 }
 return minAns;
}
```

Rowise sum

Row wise

		0	1	2	3	
↑	0	10	20	5	7	→ 42
row	1	2	4	6	8	→ 20
↓	2	10	15	15	10	→ 50

← col →

Code:

```
void rowwiseSum( int arr[][4],
      int row, int col ){
 for( int i=0; i<row; i++) {
     int sum=0;
   for( int j=0; ; <col ; ;j++) {
       sum = sum + arr[i][j) ;
   }
 cout<< sum<<endl;
}
```

columnwise sum

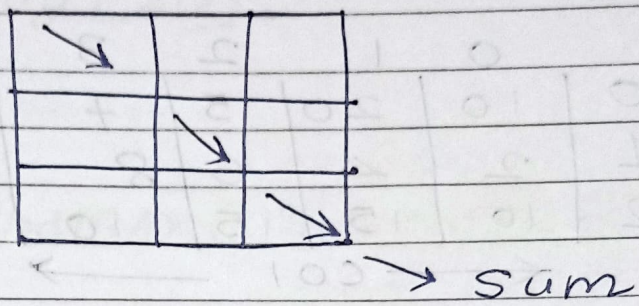
		0	1	2	3
↑	0	10	20	5	7
row	1	2	4	6	8
↓	2	10	15	15	10
		← col →			
		22	39	26	25

Code:

```
void columnWiseSum (int arr[][4],
        int row, int col ) {
For(int i=0 ; i<col; i++) {
     int sum=0
  for( int j=0; j<row; ++) {
      sum = sum + arr[j][i] ;
   }
  cout<< sum<<endl;
}
```

H.W → other diagonal

Diagonal Element Sum



Assume, Row = Col

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

Transpose of a Matrix

i/p

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

o/p

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

SWAPPING ⇒

arr[0][0] ↔ arr[0][0]
arr[1][0] ↔ arr[0][1]
arr[0][2] ↔ arr[2][0]

If we swap each element, then it will convert to its original form again. To avoid this we will make small change in the code.

Code:

```
void transpose (int arr[][4], int row,
                int col) {
    for (int i=0; i<row; i++) {
        for (int j=i; j<col; j++) {
            swap (arr[i][j], arr[j][i]);
        }
    }
```

we can use lower triangle also.

Vector in 2D.

1D → vector<int> arr

2D → vector<vector<int>> arr

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

- vector<vector<int>> → 2D Array
- arr → Name
- 5 → row size
- vector<int>(0,0) → EK row mai 10 column hai 0 se initialized Sab

What is Jagged Array?

2-D Array having different column size at each row.