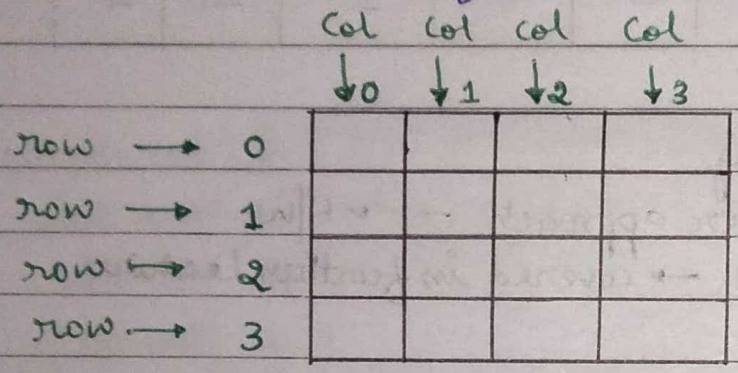
ARRAY [Level-3]

Date 11/09/23

2-D-Array :- 2D-Structure which contain rows and columns its similar to matrix.

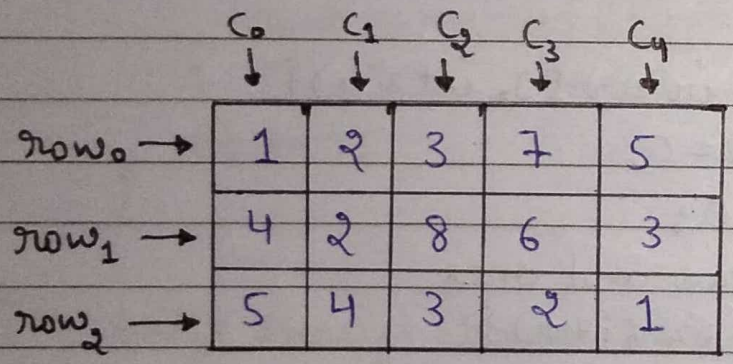
Creation of array :-

Ex :- int arr [No. of Rows] [No. of Columns]



Initialisation :-

Ex :- 3 rows, 5 cols → int arr[3][5] = {
{1, 2, 3, 7, 5},
{4, 2, 8, 6, 3},
{5, 4, 3, 2, 1}
}



☆ Specify no. of colums for initialising 2-D array

Access array values in 2-D Array :-

Syntax = array_name [Row_Index] [Column_Index]

Ex :-

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

arr

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][0] = 70, arr[2][1] = 80, arr[2][2] = 90

Spiral

Date ..../..../.....

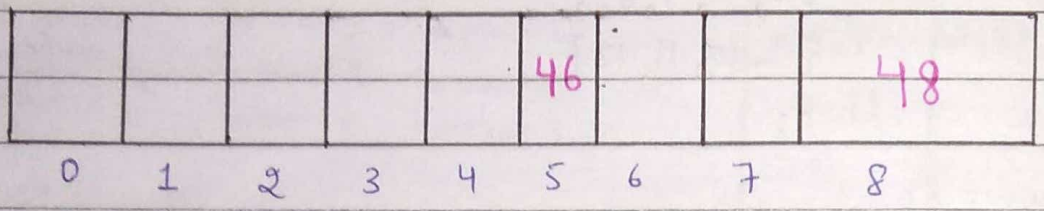
Storage of 2-D array in memory :-

We are visualising 2-D array as matrix but its actually stored as a linear array in the memory.

Ex:- int arr[3][3]

arr →

	0	1	2
0	96	25	100
1	67	52	46
2	44	60	48



↳ formula is used to convert 2-D array into linear array :-

$C \times i + j$

- C : Total no. of columns
- i : Current Row no. of that element
- j : Current column no. of that element

Ex:-

arr[2][2] is at $(C \times i + j)^{th}$ index of linear array

i.e. $3 \times 2 + 2$

$= 8^{th}$ index

arr[1][2] is at $(C \times i + j)^{th}$ index

i.e $3 \times 1 + 2$

$= 5^{th}$ index

↳ Keep in mind :- (i) 2-D array initialise karte time atleast column size specify krna mandatory hai.

(ii) 2-D array ko function mein pass krte time bhi same condition follow hogi i.e. atleast column size specifically batana mandatory hai.

Date ..../..../.....

Q Print 2-D array.

```
int main(){
 int arr[3][4] = {
                  {1, 2, 3, 4},
                  {5, 6, 7, 8},
                  {9, 10, 11, 12}.
                  };
  int row = 3;
  int col = 4;
  // function call
  printArray (arr, row, col);
  return 0;
}

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

          cout << arr[i][j] << " ";
      }

      cout << endl;
    }
}
```

Output

1 2 3 4
5 6 7 8
9 10 11 12

∴ Row wise access

Date ..../..../.....

→ Column wise access

for each column, every row is printing.

Col = 0
- ↳ row = 0
- ↳ row = 1
- ↳ row = 2
- ↳ row = 3

Col = 1
- ↳ row = 0
- ↳ row = 1
- ↳ row = 2
- ↳ row = 3

Col = 2
- ↳ row = 0
- ↳ row = 1
- ↳ row = 2
- ↳ row = 3

	0	1	2
0	(0,0) 1	(0,1) 2	(0,2) 3
1	(1,0) 10	(1,1) 20	(1,2) 30
2	(2,0) 11	(2,1) 21	(2,2) 13
3	(3,0) 20	(3,1) 22	(3,2) 53

Outer loop → i=0 to i<col

↳ Inner loop → j=0 to j< row

Printing → arr[col][row]

Code

```
void colWisePrint(int arr[][3], int row, int col){
    for(int i=0; i<col; i++){
        for(int j=0; j<row; j++){
            cout<< arr[j][i]<<" ";
        }
        cout<<endl;
    }
int main(){
    int arr[4][3] = {
                        {1,2,3},
                        {10,20,30},
                        {11,21,13},
                        {20,22,53}
                    };
    int row=4; int col=3;
    colWisePrint(arr, row, col);
    return 0;
}
```

Output :-

```
1 10 11 20
2 20 21 22
3 30 13 53
```

Spiral

Date ..../..../.....

Q Search the value of target in given array and return true if target present and return false if target is absent.

target = 70

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

Code :-

```
bool findTarget (int arr[][3], 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) {
              return true;
          }
      }
  }
  // Iss line par tabhi aa sakte ho, jab saare element check
  // ho chuke or target nhi mila hoga
  // return krdo false;
  return false;
}
int main (){
  int arr[][3] = { {10, 20, 30},
                   {40, 50, 60},
                   {70, 80, 90}
                 };
  int row = 3;
  int col = 3;
  int target = 70;
  // Function Call
  Cout << " found or Not : " << findTarget (arr, row, col, target);
  return 0;
}
```

Output :-

Found or Not : 1

Spiral

Date ..../..../.....

Q find maximum no. in an array.

Initialise answer variable with INT_MIN and then compare it with all elements. If "no." > maxAns then store that "no." in maxAns. Traverse array using 2 loops

Code:-

```
int findMaxInArray(int arr[][3], int row, int col){
    int maxAns = INT_MIN; //Initialisation with INT_MIN
    for(int i=0; i<row; i++){
        for(int j=0; j<col; j++){
            if( arr[i][j] > maxAns){ //comparison
                maxAns = arr[i][j]; //updation
            }
        }
    }
    return maxAns;
}
```

Q find minimum no. in an array.

Initialise answer variable with INT_MAX and traverse the array if any element found less than answer variable then store it in answer.

Code:-

```
int findMinInArray(int arr[][3], int row, int col){
    int minAns = INT_MAX; //Initialisation with INT_MAX
    for(int i=0; i<row; i++){
        for(int j=0; j<col; j++){
            if( arr[i][j] < minAns){ //comparing
                minAns = arr[i][j] //updating it with arr[i][j]
            }
        }
    }
    return minAns;
}
```

Spiral

Date ..../..../.....

Q Find row wise sum in an array.

Traverse the 2-D array row wise and initialise sum = 0 for each iteration of outer loop. Store the sum of each element with sum itself and store it into sum variable and then print sum.

Code:-

```
void rowWiseSum(int arr[][3], int row, int col){
    for(int i=0; i<row; i++){
        int sum = 0;
        for(int j=0; j<col; j++){
            sum += arr[i][j];
        }
        cout<<"Sum of row "<<i<<" is "<<sum<<endl;
    }
}

int main(){
    int arr[][3] = {{1, 2, 3},
                    {40, 5, 60},
                    {7, 0, 2}
                   };
    int row = 3;
    int col = 3;
    rowWiseSum(arr, row, col);
    return 0;
}
```

Output:-

```
Sum of row 0 is 6
Sum of row 1 is 105
Sum of row 2 is 9
```

Q Find column wise sum in array. HW

Traverse the array column wise and initialise sum with 0. for each column add every current row element in sum variable then print the sum when all rows are traversed.

Date ..../..../.....

Code:-

```
void colWiseSum (int arr[][3], int row, int col) {
    for (int i=0; i< col; i++){
        int sum=0;
        for (int j=0; j< row ; j++) {
            sum += arr[j][i];
        }
        cout << "sum of col " << i << " is " << sum << endl;
    }
}
int main(){
    int arr[][3] = { {1, 2, 3},
                     {40, 5, 60},
                     {7, 0, 2}
                   };
    int row = 3, col = 3;
    colWiseSum (arr, row, col);
    return 0;
}
```

Output:-

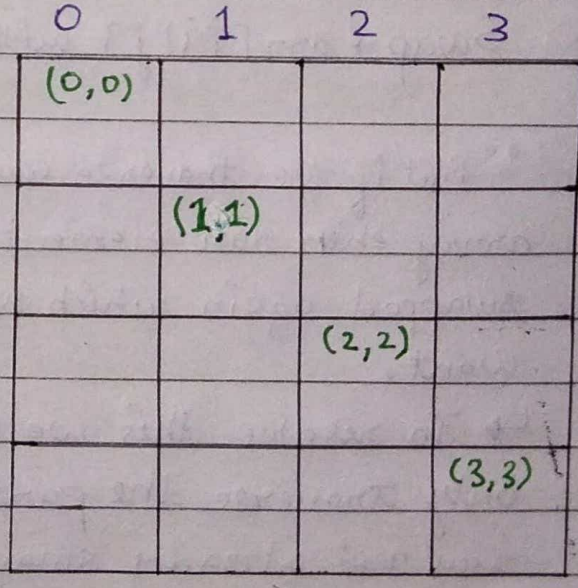
Sum of Col 0 is 48
Sum of Col 1 is 7
Sum of col 2 is 65

Q Print diagonal of 2-D array. Assuming no. of rows = no. of columns.

∵ for this diagonal we have elements whose row and column index are same.

Code:-

```
void diagonalPrint(int arr[][3], int row){
    for (int i=0; i< row; i++){
        cout << arr[i][i] << " ";
    }
}
```



Spiral

Date ..../..../.....

Q Print diagonal sum of 2-D array. Assuming no. of rows = no. of columns.

∴ Initialise a sum variable with 0.

↳ Then traverse the array and add only those elements in sum whose row and column index are the same.

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

Code:-

```
void diagonalSum (int arr[][3], int row)
  {
      int sum = 0;
      for ( int i = 0; i<row; i++ ){
          sum += arr[i][i];
      }
      cout << sum;
  }
```

Imp

Q Print transpose of 2-D matrix. Assuming no. of rows = no. of columns.

Transpose means i^{th} row becomes i^{th} column.

Approach:-

Swap → arr[i][j] with arr[j][i]

∴ But if we traverse complete array then our elements gets swapped again which we don't want.

↳ To resolve this we should only traverse the part which was not already swapped

↳ Inner Loop will initialised with j==i, so that we can skip the swapped column.

	0	1	2	3
0	(0,0) 10	(0,1) 20	(0,2) 30	(0,3) 40
1	(1,0) 5	(1,1) 4	(1,2) 3	(1,3) 2
2	(2,0) 1	(2,1) 7	(2,2) 19	(2,3) 22
3	(3,0) 28	(3,1) 6	(3,2) 31	(3,3) 9

Spiral

Date ..../..../.....

Code:-

```
void transpose (int arr[][3], int row, int col) {
    // Outer Loop
    for(int i= 0; i< row; i++) {
    // Inner Loop → from j==i to j< col
        for(int j=i; j< col; j++){
            swap( arr[i][j], arr[j][i]);
        }
    }
}
```

Vector :-

A vector stores elements of a given type in a linear arrangement and allow fast random access to any element.

↳ Syntax to create a vector :-

```
vector<data_type> vector_name(size);
```

Ex :-

```
vector<int> v(5);
```

↳ To check the size of vector :-

size() - It's a function which return the size of vector.

Ex :-

```
cout << v.size(); // It will print the size of 'v' vector i.e 5
```

* By default, all vector elements are assigned with '0'.

↳ To initialise vector elements :-

Ex :-

```
vector<int> arr(5, 40);
```

∴ Here, all the elements of vector 'arr' will initialise with '40'

Date ..../..../.....

↳ Insert elements into 1-D vector :-

We can use inbuilt function push_back() to push any value.

Ex:- arr.push_back(1);
arr.push_back(2);

∴ This will push/insert 1 and 2 in the vector named "arr"

2-D vector :-

↳ Syntax to create 2-D vector :-

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

int → Datatype
space
arr → vector_name

↳ Visualisation of 1-D and 2-D vectors :-

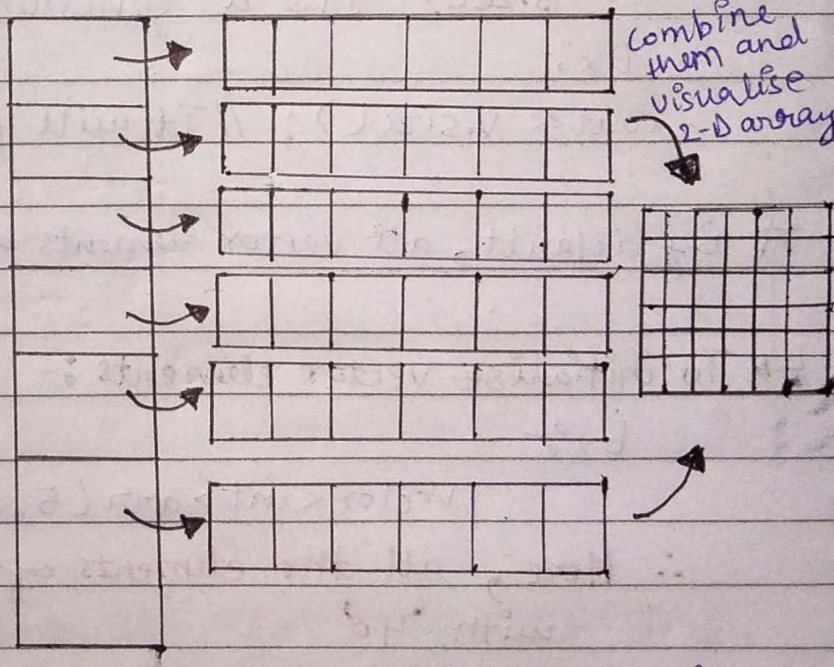
Ex:-

vector <int> v

1D | 2 | 4 | 6 | 8 | 9 | 10 |

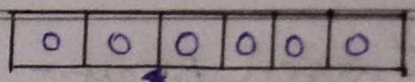
↳ 1-D array which have integer values in it.

vector <vector <int> >



∴ A 1-D array contain <vector<int> > type of elements in each place

Date ..../..../.....



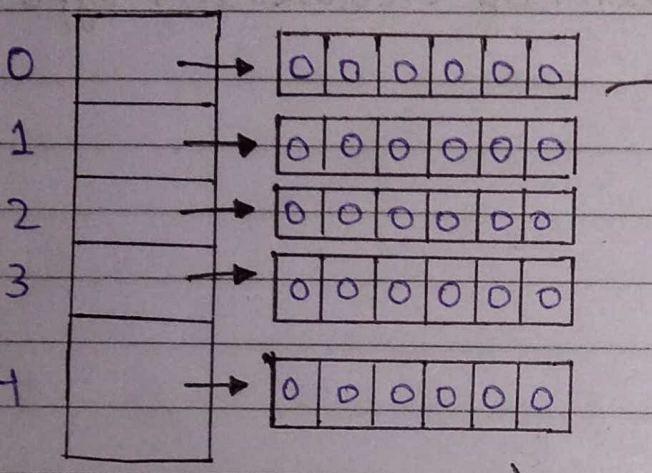
```
Vector < vector <int> > arr ( 5 , vector <int> (6,0) )
```

2-D array

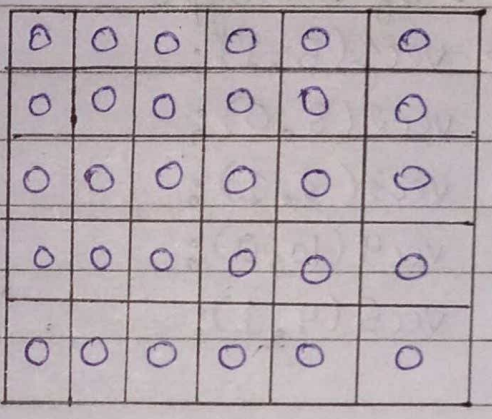
name

Row size

Row Item
↳ Initialise → with a vector of size 6 that is initialised with 0



1-D array with size '5' and initialised with vector of integer type



5

6

∵ A 2-D array with 5 rows and 6 columns which is initialised with zeros

Code:-

Row size ← Each Row item initialise with this vector → And this vector have size = 10 and initialise with 0.

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

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

Output:-

```
10 10 10 10 10 10 10 10 10 10
10 10 10 10 10 10 10 10 10 10
10 10 10 10 10 10 10 10 10 10
10 10 10 10 10 10 10 10 10 10
10 10 10 10 10 10 10 10 10 10
```

Spiral

Date ..../..../.....

Jagged Array -

A 2-D array which has different no. of columns in its each row.

Code:-

```
int main(){
    vector< vector<int> > brr;
// This line above will make a 2-D vector of <vector<int>> type

// 1-D vectors of <int> type
  vector<int> vec1(5,1);
  vector<int> vec2(8,0);
  vector<int> vec3(3,1);
  vector<int> vec4(10,0);
  vector<int> vec5(4,1);

// Inserting 1-D vectors in 2-D vector
  brr.push_back(vec1);
  brr.push_back(vec2);
  brr.push_back(vec3);
  brr.push_back(vec4);
  brr.push_back(vec5);

// Printing 2-D vector
  for(int i=0; i<brr.size(); i++){
    for(int j=0; j<brr[i].size(); j++){
        cout<<brr[i][j]<<" ";
    }
    cout<<endl;
  }
return 0;
}
```

Output:-

```
1 1 1 1 1
0 0 0 0 0 0 0 0
1 1 1
0 0 0 0 0 0 0 0 0 0
1 1 1 1
```

Spiral