* 2-D Arrays: 2-Dimensional Arrays

↓

It is a Data structure collection of multiple arrays or grids. 2D contains different arrays in the forms of grid.

Ex: int arr[] = {1, 2, 3, 4} → Ex of 1-D Array.

1	2	3	4

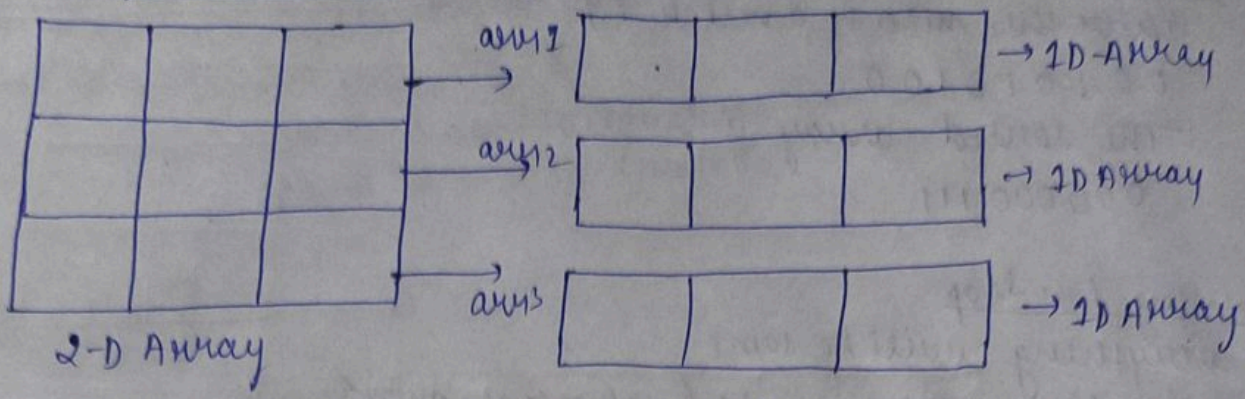
2-D Array

row 0	1	2	3
	4	5	6
	11	9	7
	10	12	8

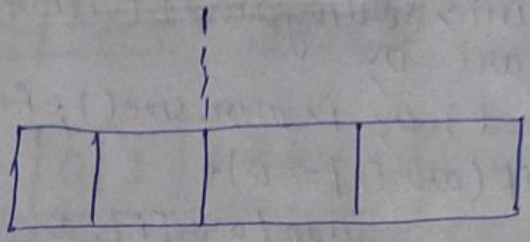
it is an example of 2-D Arrays.

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

2-D Arrays are collection of multiple 1-D Arrays:



It is easy to create 1 array 2 array but won't be possible to create 1000's arrays.



Here 2-D Arrays comes in practice.

* Syntax of 2-D

~~data~~ int arr[100][100]

no. of colms.

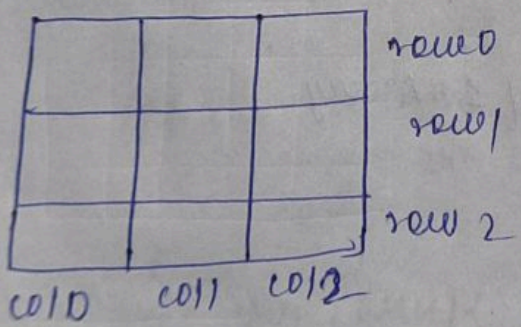
datatype variable — no. of row

ex:

int arr[3][3]

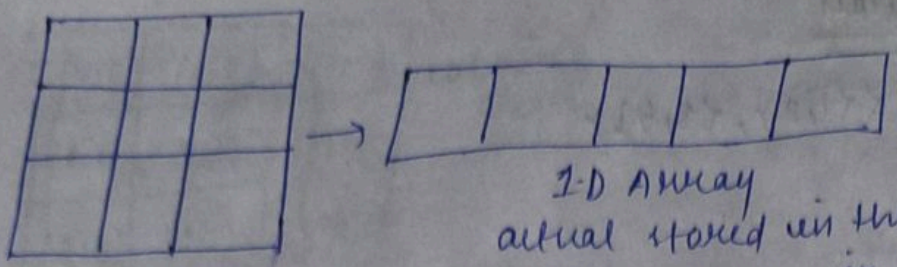
rows = 3
cols = 3

Total elements = 3×3 = 9



* How 2-D is stored inside memory

2D array stored in one memory location but in memory it is not stored in the form of 2D in memory it is stored in the form of 1-D array. We only visualize the 2-D array but actually it is stored in the form of 1-D array in memory.



2-D array visualize

1-D Array actual stored in the form of 1-D array in memory.

Ex:

	col0	col1	col2
row0	5	7	9
row1	12	4	2
row2	1	6	7

0	1	2	3	4	5	6	7	8
5	7	9	12	4	2	1	6	7

1-D actual stored in this form

visualize

r = no. of rows in 2-D array
c = no. of col in 2-D array.
i = i^{th} row
j = j^{th} col

To find at what memory an 2D-Array element is stored in 1-D continuous memory location. Given by formula.

⇒ $c * i + j$ → j^{th} col
(c: no. of cols, i: i^{th} row)

∴ arr[1][2] is stored in memory at c=3, i=1, j=2
$c*i+j = 3\times1+2 = 5^{th}$ index of 1-D array.

∴ arr[2][2] is stored in memory at c=3, i=2, j=2
$c*i+j = 3\times2+2 = 8^{th}$ index of 1-D array

∴ arr[2,1] is stored in memory at c=3, i=2, j=1
$c*i+j = 3\times2+1 = 7^{th}$ index of 1-D array

How to access 2-D array

int arr[i][j]
i: row index, j: cols index.

* Creation: Initialization

```
int arr[2][2] = {{1,2},{3,4}};
```

	0	1
0	0	2
1	3	4

* Taking Input and giving output in 2-D Array

In 1-D Array

```
cin >> arr[i]; //input
cout << arr[i] << " "; //output.
```

In 2-D Array

```
cin >> arr[i][j]; //input.
cout << arr[i][j]; // output
```

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

arr[0][1] = 2
arr[2][1] = 8
arr[1][2] = 6

m cols →

n rows ↓

n rows : → 0 - n-1
m cols : → 0 - m-1

* Row-wise access: can be done by nested array.

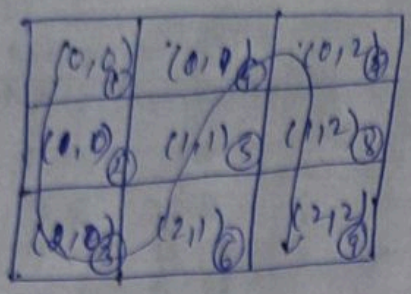
	0	1	2
0	(0,0) ①	(0,1) ②	(0,2) ③
1	(1,0) ④	(1,1) ⑤	(1,2) ⑥
2	(2,0) ⑦	(2,1) ⑧	(2,2) ⑨

row-wise access

0th row → ⓪col ①col ②col
1th row → ③col ④col ⑤col
2nd row → ⑥col ⑦col ⑧col

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

size of row
size of col
rows cols



column wise access

0th col → 0th row 1st row 2nd row
1st col → " " "
2 " " → " " "

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

just row-col
cout<<arr[i][j]; &
in col-col
cout<<arr[j][i];
replace or swap i and j

```
for(int i=0; i<3; i++){
    for(int j=0; j<3; j++){
        cout<<arr[j][i]<<" ";
    }
}
```

for i=0	for i=1;	for i=2;
j=0	j=0	j=0
print arr[0][0]	print arr[0][1]	print arr[0][2]
j=1	j=1	j=1
print arr[1][0]	print arr[1][1]	print arr[1][2]
j=2	j=2	j=2
print arr[2][0]	print arr[2][1]	print arr[2][2]

→ Declare: int arr[1][2];

→ Initialization: int arr[2][3] = {{1,2,3},{4,5,6}};

→ print 2D Array:
```
for(int i=0; i<row; i++){
    for(int j=0; j<col; j++){
        cout<<arr[i][j]<<" ";
    }
}
```

→ input 2D Array: same
cin>>arr[i][j];

① Row sum print:

	0	1	2	
0	1	2	3	= 6
1	4	5	6	= 15
2	7	8	9	= 24
3	10	11	12	= 33

```
void rowsum (int arr[][30] ,int rows, int cols){
    for(int i=0 ; i<rows; i++){
    int sum=0;
    for(int j=0 ; j<cols; j++){
        sum+ = arr[i][j];
    }
    cout<<"The sum is: "<< sum <<endl;
}
}

int main(){
    int rows, cols;
    cout<<" Enter no of rows: "; cin>>rows;
    cout<<" enter no of cols: "; cin>>cols;
    int arr[30][30];
    for(int i=0; i<rows; i++){
        for(int j=0 ; j<cols; j++){
            cin >> arr[i][j];
    }
}
for( int i=0; i<rows; i++){
    for(int j=0 ; j<cols ; j++){
        cout << arr[i][j] <<" ";
    }
    cout<<endl;
}
cout<<endl;
rowsum( arr, rows, cols);
```

Output

```
No. of rows : 4
No. of columns : 4
1 2 3 4
5 6 7 8
9 10 11 12
13 14 15 16
The sum is 10
The sum is 26
The sum is 42
The sum is 58
```

② Linear search 2-D Array

```
bool findelement (int arr[][30], int rows, int cols, int ele){
    for (int i=0 ; i<rows ; i++){
        for (int j=0 ; j< cols ; j++){
            if (arr[i][j] == ele){
                return true;
            }
        }
    }
    return false;
}

int main (){
    //same as row-wise sum
    int element;
    cout << " enter no to find :"; cin >> element;
    if (findelement (arr, rows, cols, element)){
        cout << " Element Found";
    }
    else {
        cout << " Not Found";
    }
```

Output

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

element = 11
Element found

(3) Maximum No. Found in 2D Array:

```
int findmaxno (int arr[][30], int rows, int cols){
    int max = INT_MIN;
    for (int i=0; i<rows; i++){
        for (int j=0; j<cols; j++){
            if (arr[i][j] > max){
                max = arr[i][j];
            }
        }
    }
    return max;
}
int main(){
    // same as row-wise sum
    int max = findmaxno(arr, rows, cols);
    cout << "The max no"; cout << max;
```

Output

```
1   12  16  90
81  43  28  144
91  100 128 113
14  23  21  20
```

Max no: 144.

(4) Minimum No Found in 2-D Array:

Everything will be same condition will change

```
int min = INT_MAX;
if (arr[i][j] < min){
    min = arr[i][j];
}
}
```

(5) transpose of 2D Array.

```
void transpose (int arr[][30], int rows, int cols){
    int temp;
    for (int i=0; i<rows; i++){
        for (int j=0; j<cols; j++){
```

```
            temp = arr[i][j];
            arr[i][j] = arr[j][i];
            arr[j][i] = temp;
        }
    }
    cout<<endl;
    for(int i=0; i<rows; i++){
        for(int j=0; j<col; j++){
            cout<< arr[i][j]<<" ";
        }
    }
}
int main(){
    // same of row-wise sum.
    transpose(arr, rows, cols);
}
```

output

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

Transpose

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

* Why we need to pass the array[][ ] second parameter during the definition of function.

i.e
```
int func( int arr[][30]){
    // code
}
int main(){
    int arr;
    func(arr);
}
```

↳ we need to pass the 2nd parameter of an 2D array, during defining the function bcz as we visualize the 2D array but actually an 2D array is stored as 1-D contigous memory location. To find at what memory an 2-D array is stored we need columns i.e $c*i+j$

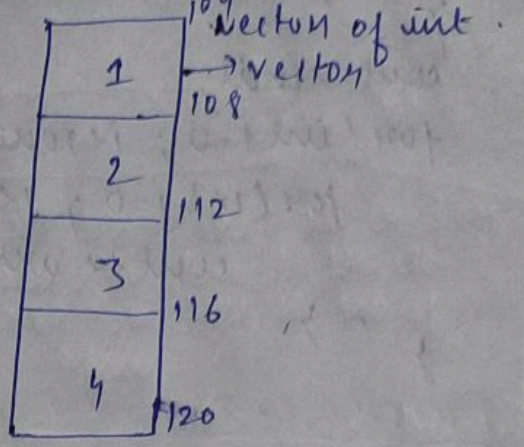
↓ formula of finding at what memory it is stored

where c = no. of column i = ith row & j = jth row.

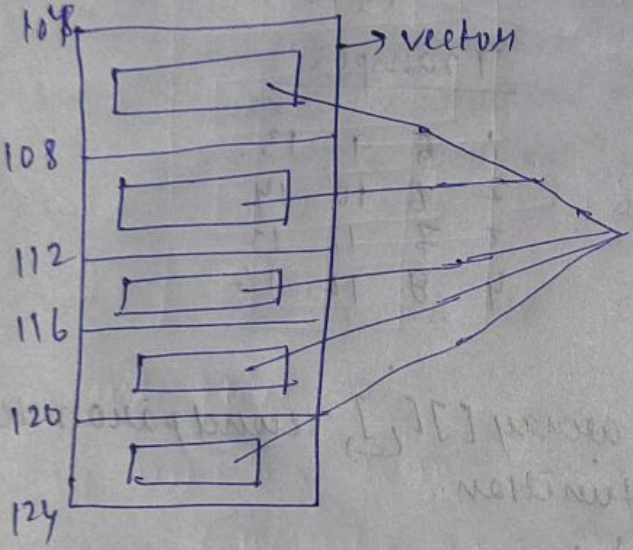
* Vectors in 2D : It is a Dynamic array whose size can change during runtime.
It is a Data structure.

In 1-D Array we generally use vector in the form

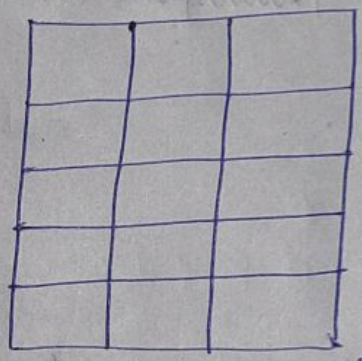
vector<int> arr
↓ ↓ ↓ variable name.
name datatype



In 2-D array its generally vector of vector means nested vectors



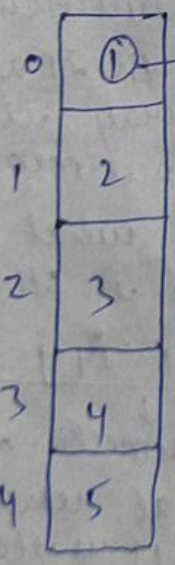
=



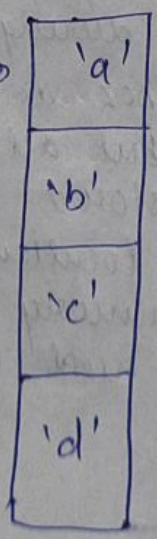
nested vectors

nested vectors resulted in formation of 2D Arrays.

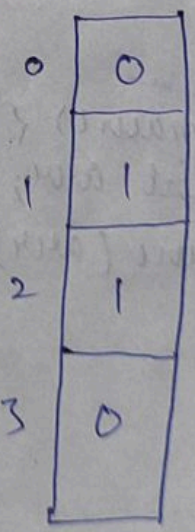
* Vectors of int
vector<int> arr;



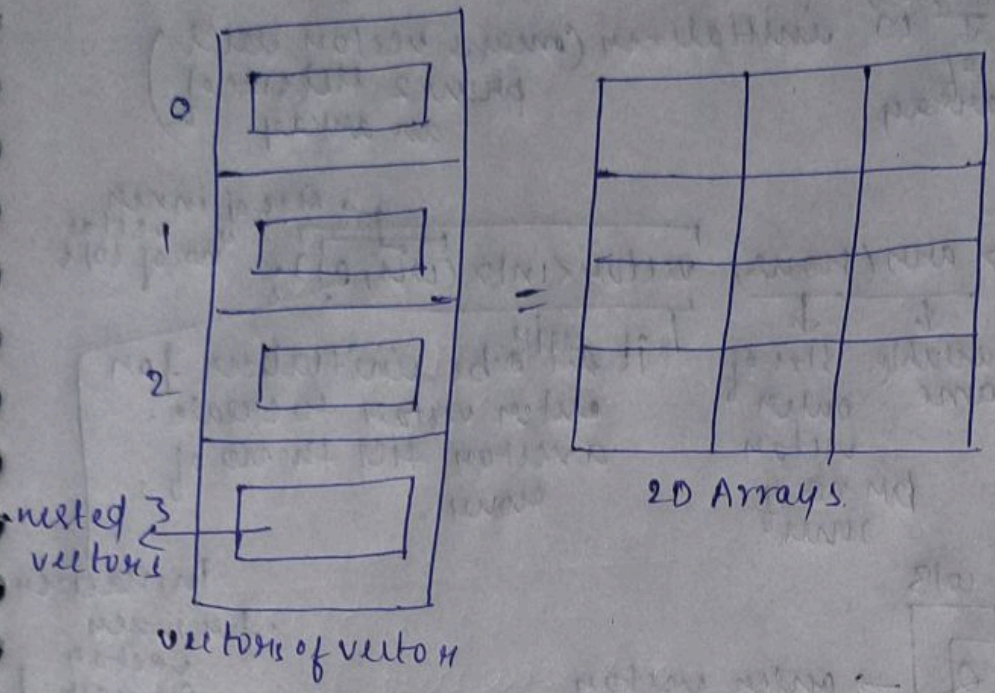
vector of char
vector<char> arr;



vector of bool
vector<bool> arr;



In 2-D Arrays it is vector of vector i.e nested vector.



* Declaration of 2-D vector:

In 1-D array.

vector<int> arr(n, 2);
↳ initializer
↳ size of an array

In 2-D Array

vector<vector<int>> arr;
arr[][];

↓ outer vector
↓ * inner vector
↓ datatype
↳ variable name.

→ How we can write 2D vector if we want the rows & cols from user & initialize.

vector<vector<int>> arr (rows, vector<int> (cols, 0))

↓ nested vector
↓ outer vectors no. of rows
↓ inner vectors no. of cols
↓ initialization

* Lets understand it in depth:

→ In 1-D array

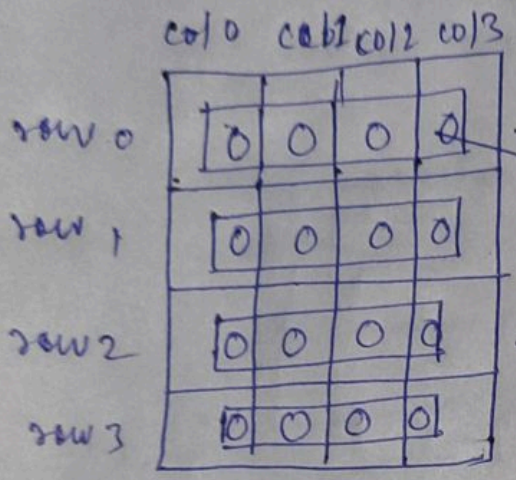
vector<int> arr(n, 2) ↳ initializer (means vector will print 2 till size of an array)

↓ size of an array

→ In 2-D Array

vector<vector<int>> arr(rows, vector<int>(cols, 0));

- vector<vector<int>> → indicates will create 2-D Array.
- arr → variable name
- rows → size of outer vector OR no. of rows.
- vector<int>(cols, 0) → it will be initializer for outer vector to create a vector till the no. of rows.
- cols → size of inner vector no. of cols
- Initializer for inner vector i.e nested vector



to print 2D vector

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

arr.size() → size of outer vector / no. of row

arr[i].size() → size of inner vector / no. of columns.

Here a[i].size() : shows that at what particular rows there are how many columns