12/02/2023

Till now we have studied about 1D arrays. Today we will be studying about the 2D arrays.

2D Arrays

int arr [ ] = {1, 2, 5, 7} ;

arr →

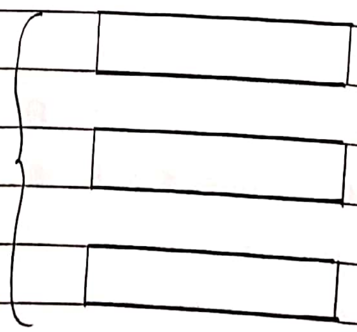
1	2	5	7

This is how 1D arrays are stored in the memory. But the question is what happens in case of 2D arrays. We can relate 2D arrays with Tic-tac-toe game which have 9 cells.

	col 0	col 1	col 2
row 0			
row 1			
row 2			

A grid is formed where there are multiple rows and multiple columns. The above grid can be formed by taking 3, 1D arrays of size 3 each.

3, 1D arrays



But this is a small grid, that's why we created 1D arrays but what if there are 1000 rows & it is not practical to create 1000 1D arrays. Hence 2D arrays make our job simpler.

```
         ┌ name              ┌ no. of columns
int   arr [1000][1000];
↑               └ no. of rows
data type
```

Also we can mention different no. of rows & different number of columns.

Memory representation of 2D arrays

int arr[2][2]; → Total elements = 2 × 2 = 4

arr → | 1 | 2 | 3 | 4 | → 2D array

Matrix was just used to visualize the 2D arrays but in memory, the 1D array representation is used.

	col0	col1
row0	1	2
row1	3	4

} Just way of visualizing 2D arrays

Accessing elements of 2D arrays

In the above 2D array, say we want to access 3. We can access it by $\underbrace{\text{arr [1] [0]}}_{3}$

arr [0] [0] → 1
arr [0] [1] → 2
arr [1] [1] → 4

arr [ ] [ ]
row index → column index

Formulae for mapping in 1D array representation is $C * i + j$

$C \rightarrow$ no. of columns

$i \rightarrow$ row-index

$j \rightarrow$ column-index

arr[0][1] = 2

Formulae = $2 \times 0 + 1 = 0 + 1 = 1$st index in 1D array.

1	2	3	4
0	1	2	3

2 is present at 1st index in 1D array.

arr[1][1] = 4

Formulae = $2 \times 1 + 1 = 2 + 1 = 3$rd index, 4 is stored in 1D array.

Note $\rightarrow$ arr[0][1] $\neq$ arr[1]. It is just a way that in memory 1D array is used but the actual interface is the matrix only so use arr[i][j] to access the elements of 2D array.

Initialization of 2D array

int arr[2][2] = $\{ \underbrace{\{1,2\}}_{\text{row-0}}, \underbrace{\{3,4\}}_{\text{row-1}} \}$;

	col-0	col-1
row-0 →	1	2
row-1 →	3	4

Input and Output in 2 D Array

Input → cin >> arr [i][j];

Output → cout << arr [i][j];

We always have to play within 0 to n-1 index where n → rows and colums = n.

Note → arr [i][j]

No. of rows = n

No. of columns = m

$i \in [0, n-1]$
$j \in [0, m-1]$ } Never go out this range

Accessing elements row-wise

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

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

1	2	3
4	5	6
7	8	9

arr

Output

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

Accessing elements column-wise

```
for(int i=0 ;i<3; i++) {
```

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

(swapped in case of column wise)

Output

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

Similarly we can take row-wise input & column-wise input.

Problem Solving

1) Row sum print

i/p →

1	2	3
4	5	6
7	8	9

o/p →
6
15
24

Go to every row, calculate the sum & then print. We have to simply do row wise traversal.

Code

```
Void printRowSum (int arr [][3], int rows
                  int cols) {
```

```
for (int i=0 ; i< rows ; i++) {
        int sum = 0 ; → After each row , sum = 0
        for (int j = 0 ; j < cols ; j++) {
                sum = sum + arr [i] [j] ;
        }
        cout << "Row" << i << "sum is"
        << sum << endl ;
}
```

Note → While passing 2D arrays in function, we need to mention the bound of column. It is a rule. Formulae of $c * i + j$ is the reason behind this.

2) Column sum print

i/p → Same array as that of Ques-1
O/p → 12
15
18

We just have to do column-wise traversal & print the sum just like we did in row wise sum print.

In the above code just change
sum = sum + arr [i] [j] to
sum = sum + arr [j] [i]

3) Linear search in 2D array

i/p → Same array as that of ques-1
element = 3
o/p → True

We can do either row-wise traversal or column-wise traversal & compare each element with the element that is given to us as in i/p. If element is matched return true else return false if not found even after traversing full array.

Code

```
bool
    findkey (int arr[][3], int r, int c, int
        for(int i=0; i<r; i++) {
            for(int j=0; j<c; j++) {
                if (arr[i][j] == k) {
                    return true;
                }
            }
    } return false;
}
```

4) Maximum and minimum in 2D array.

i/p → same array as that of ques-1
o/p →
Maximum = 9
Minimum = 1

Initialize maxi = INT_MIN & mini = INT_MAX and compare maxi & mini with each element.

✓ maxi < arr[i][j] → update maxi
✓ mini > arr[i][j] → update mini

After traversing full array, return the maxi & mini

Code

```
int maximum (int arr[][3], int r, int c) {
        int maxi = INT_MIN;
        for (int i=0 ; i<r    ; i++) {
            for (int j=0 ;  j<c ; j++) {
                if (arr[i][j] > maxi) {
                    maxi = arr[i][j];
                }
            }
        } return maxi ;
}

int minimum (int arr[][3], int r, int c) {
        int mini = INT_MAX;
        for (int i=0; i<r; i++) {
            for (int j=0; j<c; j++) {
                if (arr[i][j] <mini) {
                    mini = arr[i][j];
                }
            }
        }
    return mini;
}
```

5) Transpose of matrix

i/p → same array as that of ques-1

o/p →

1	4	7
2	5	8
3	6	9

We just have to change the rows into the column. We need to perform the swap on each cell.

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

But the inner for loop won't run fully as some elements will be swapped twice & would reach to the original position and hence the matrix would be same.

Code

```
void transpose (int arr [][3], int r , int c){

        for (int i=0 ; i<r ; i++){
            for (int j = 0 ; j<i ; j++){
                swap (arr [i][j], arr [j][i]);
            }
        }
}
```

Better way to use 2D arrays

1 D array → vector <int> arr ;

2D array → We will be using vector of vector concept

→ vector of vector having 3 cells

outer vector
inner vector

```
vector <vector <int>> v;
```

data type ; name of vector

Each block will contain vector of integers.
How to mention no. of rows & no. of columns?

```
vector <vector <int>> arr (3, vector<int> (5,0));
```

no. of rows = 3 ; no. of columns = 5

Inserting data in the 2D vector / vector of vector.

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

In arr, vectors will be pushed.

```
vector <int> a {1,2,3};
arr.push_back (a);
```

No. of rows in vector of vector
arr.size() will tell the no. of rows in vector of vector
No. of columns in vector of vectors
arr[i].size() where i is row-index will tell the no. of columns in each row.

Note →

```
vector <vector <int>> arr (rows,
                          vector <int> (col,0));
```

Inner vector will be initialized to 0.