

Matrix / 2D-Array

☰ Tags	Algorithm Data-Structure MyNotes dynamic_programming
🕒 Created time	@November 8, 2022 6:21 PM
🕒 Last edited time	@November 11, 2022 7:54 AM
⚙ Status	Completed
🔗 URL	
🔗 URL 1	

Array is a datatype collection of same datatype in a contiguous location. these datatypes can be literally any datatype primitive(int,char,double.etc) or user-defined and since array is a data type itself we can make a array of an array and when array is consist of array its called a “2D-Array”. it can be visualised as a matrix, grid, or a table of data. and its not just a limit there since its an array of array of some data type it can be also stores a array in that array which will make it an array of array of array or 3d array we can go as much as we want. though any thing above is can’t be visualised in real world but it is possible in to make a multi-dimension array.

Declaration and visualisation -

we declare a 1d array just like any other datatype (i.e. first mention the datatype followed by name of the variable ex. int name;) with just a slight addition to it a pair of square bracket with size mentioned inside in it (int x[10];) to make it a multiple

dimensional array we just treat the array-name[size] as the variable name and add a square bracket in front of it (array-name[size][size-2]) so the number inside first square bracket is the number of array inside that the array and second denotes the number of elements inside the array. for visualising it as a matrix we can interpret it as row and column where number in first square bracket is row and second as column.



The diagram shows the C++ declaration `int arr[4][5];` on a dark background. Two white arrows point from the numbers in the brackets to labels below. The first arrow points from the '4' to the label `# rows`. The second arrow points from the '5' to the label `# columns`.

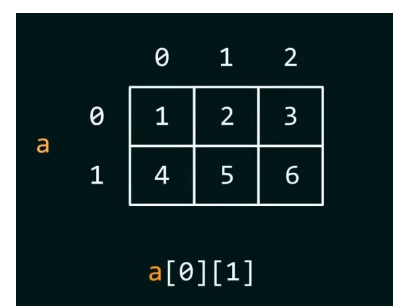
Initialisation and access -

to initialise(hard-code) any array we have to list all the element inside a curly bracket separating by commas ex. {element-1,element-2,element-3,element-4,element-5} and initialisation a 2 dim array is same elements inside curly it just look a little different but all it is arrays inside array

```
//initilision 1d array
int arr[]{1,2,3};
//2d
int arr[][]{ {1,2,3} , {2,3,4} };
```

example:

to access a element any perticular element in array we have to add a square bracket just like initialization but this time with inside the index of element we want like `arr[2]` will return the 3rd element of array and `arr[0][1]` will return the first array's 2nd element of the multi-dim array.



The diagram shows a 2D array named 'a' represented as a matrix. The rows are indexed 0 and 1, and the columns are indexed 0, 1, and 2. The matrix contains the following values:

	0	1	2
0	1	2	3
1	4	5	6

Below the matrix, the expression `a[0][1]` is shown, indicating the value 2 at the first row and second column.

Printing 2d array

row wise

```

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

```

Some questions on 2D

Linear search

▼ code

```

pair search(arr[][5],int x,int row,int col)//it is important to add size of array
{
    for(int i = 0;i < row;i++)
        for(int j = 0;j < col;j++)
            if(arr[i][j] == x) return {i,j};
    return {-1,-1};
}

```

Binary search I

▼ code

```

bool searchMatrix(vector<vector<int>>& matrix, int target) {
    int start = 0;
    int col = matrix[0].size();
    int end = (matrix.size() * matrix[0].size()) -1 ;
    while(start <= end){
        int mid = start + (end - start)/2;
        int element = matrix[mid/col][mid%col];
        if(element > target) end = mid-1;
        else if(element < target) start = mid+1;
        else return true;
    }
    return false;
}

```

Binary search II

▼ code

```

bool searchMatrix(vector<vector<int>>& matrix, int target) {
    int row = matrix.size();
    int col = matrix[0].size();
    int x = 0;
    int y = col-1;
}

```

```

        while(x < row && y >= 0){
            if(matrix[x][y] == target) return true;
            else if(matrix[x][y] > target) y--;
            else x++;
        }
        return false;
    }
}

```

Rotate array

▼ code

```

vector<int> wavePrint(vector<vector<int>> arr, int nRows, int mCols)
{
    vector<int> answer;
    for (size_t i = 0; i < mCols; i++)
    {
        if(i&1){
            for (int j = nRows-1; j >= 0; j--)
            {
                answer.push_back(arr[j][i]);
            }
        }
        else {
            for (size_t j = 0; j < nRows; j++)
            {
                answer.push_back(arr[j][i]);
            }
        }
    }
    return answer;
}

```

Spiral print

▼ code

```

vector<int> spiralOrder(vector<vector<int>> &matrix)
{
    vector<int> answer;
    int row = matrix.size();
    int col = matrix.at(0).size();
    int total = row * col;
    int rowStart = 0;
    int colStart = 0;
    int rowEnd = row-1;
    int colEnd = col-1;
    int printed{0};
    while(printed<total){
        //right
        for(int i = colStart;i <= colEnd && printed<total;i++){

```

```

        answer.push_back(matrix[rowStart][i]);printed++;
    }
    rowStart++;
    //down
    for(int i = rowStart;i <= rowEnd && printed<total;i++){
        answer.push_back(matrix[i][colEnd]);printed++;
    }
    colEnd--;
    //left
    for(int i = colEnd; i >= colStart && printed<total;i--){
        answer.push_back(matrix[rowEnd][i]);printed++;
    }
    rowEnd--;
    //up
    for(int i = rowEnd;i >= rowStart && printed<total;i--){
        answer.push_back(matrix[i][colStart]);printed++;
    }
    colStart++;
}
return answer;
}

```