Matrix

Multidimensional array

Fixed Size Array

```cpp
#include<iostream>
using namespace std;

int main(){
    int arr[3][2]={{10,20},
                   {30,40},
                   {50,60}};

    for(int i=0;i<3;i++)
    {
        for(int j=0;j<2;j++)
        {
            cout<<arr[i][j]<<" ";
        }
    }

    return 0;
}
```

OUTPUT

10 20 30 40 50 60

Variable Size Array

```cpp
#include<iostream>
using namespace std;

int main()
{
    int m=3,n=2;
    int arr[m][n];

    for(int i=0;i<m;i++)
    {
        for(int j=0; j<n;j++)
        {
            arr[i][j]=i+j;
        }
    }

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

    return 0;
}
```

OUTPUT

0 1 1 2 2 3

Double Pointer Array

```
#include<iostream>
using namespace std;

int main()
{
    int m=3,n=2;
    int **arr;

    arr=new int*[m];

    for(int i=0;i<m;i++)
        arr[i]=new int [n];

    for(int i=0;i<m;i++)
    {
        for(int j=0;j<n;j++)
        {
            arr[i][j]=10;

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

    return 0;
}
```

OUTPUT

10 10 10 10 10 10

Array Pointer

```cpp
#include<iostream>
using namespace std;

int main()
{
    int m=3,n=2;

    int *arr[m];

    for(int i=0;i<m;i++)
        arr[i]=new int[n];

    for(int i=0;i<3;i++)
    {
        for(int j=0;j<2;j++)
        {
            arr[i][j]=10;

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

    return 0;
}
```

OUTPUT

10 10 10 10 10 10

Array Of vectors

```cpp
#include<iostream>
#include<vector>
using namespace std;

int main()
{
    int m=3,n=2;

    vector<int> arr[m];

    for(int i=0;i<m;i++)
    {
        for(int j=0;j<n;j++)
        {
            arr[i].push_back(10);
        }
    }

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

    return 0;
}
```

OUTPUT

10 10 10 10 10 10

Vector Of vectors

```cpp
#include<iostream>
#include<vector>
using namespace std;

int main()
{
    int m=3,n=2;

    vector<vector<int>> arr;

    for(int i=0;i<m;i++)
    {
        vector<int> v;
        for(int j=0;j<n;j++)
        {
            v.push_back(10);
        }

        arr.push_back(v);
    }

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

    return 0;
}
```

OUTPUT:

10 10 10 10 10 10

Passing 2D array as a argument

Example 1:

```
#include<iostream>
using namespace std;

void print(int mat[3][2])
{
    for(int i=0;i<3;i++)
    {
        for(int j=0;j<2;j++)
            cout<<mat[i][j]<<" ";
    }
}

int main()
{
    int m=3,n=2;

    int mat[3][2]={{10,20},
                   {30,40},
                   {50,60}};

    print(mat);

    return 0;
}
```

OUTPUT:

10 20 30 40 50 60

Example 2:

```
#include<iostream>
using namespace std;

void print(int mat[][2],int m)
{
    for(int i = 0; i < m; i++)
    {
        for(int j = 0; j < 2; j++)
            cout << mat[i][j] << " ";
    }
}

int main()
{

    int mat[3][2] = {{10, 20},
                     {30, 40},
                     {50, 60}};

    print(mat, 3);

    return 0;
}
```

OUTPUT:

10 20 30 40 50 60

Example 3:

```
#include<iostream>
using namespace std;

const int R=3;
const int C=2;

void print(int mat[R][C])
{
    for(int i=0;i<R;i++)
    {
        for(int j=0;j<C;j++)
            cout<<mat[i][j]<<" ";
    }
}

int main()
{

    int mat[R][C] = {{10, 20},
                     {30, 40},
                     {50, 60}};

    print(mat);

    return 0;
}
```

OUTPUT:

10 20 30 40 50 60

Example 4:

```
#include<iostream>
using namespace std;

void print(int **arr, int m, int n)
{
    for(int i=0;i<m;i++)
    {
        for(int j=0;j<n;j++)
            cout<<arr[i][j]<<" ";
    }
}

int main()
{
    int m=3,n=2;
    int *arr[m];   //array of pointer

    for(int i=0;i<m;i++)
    {
        arr[i] =new int [n];

        for(int j=0;j<n;j++)
        {
            arr[i][j]=i;

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

    return 0;
}
```

OUTPUT:

0 0 1 1 2 2

Example 5:

```
#include<iostream>
#include<vector>
using namespace std;

void print(vector<int> arr[], int m)
{
    for(int i=0;i<m;i++)
    {
        for(int j=0;j<arr[i].size();j++)
            cout<<arr[i][j]<<" ";
    }
}

int main()
{
    int m=3,n=2;

    vector<int> arr[m];  //array of vector

    for(int i=0;i<m;i++)
    {
        for(int j=0;j<n;j++)
        {
            arr[i].push_back(i);
        }
    }

    print(arr,m);

    return 0;
}
```

OUTPUT:

0 0 1 1 2 2

Example 6:

```
#include<iostream>
#include<vector>
using namespace std;

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

int main()
{
    int m=3,n=2;

    vector<vector<int>> arr;  //vector of vector

    for(int i=0;i<m;i++)
    {
        vector<int>v;
        for(int j=0;j<n;j++)
            v.push_back(i);

        arr.push_back(v);
    }

    print(arr);

    return 0;
}
```

OUTPUT:

0 0 1 1 2 2

Matrix In Snake Pattern

```cpp
#include<iostream>
using namespace std;

const int R=4, C=4;

void printSnake(int mat[R][C])
{
    for(int i=0;i<R;i++)
    {
        if(i%2==0)
        {
            for(int j=0;j<C;j++)
                cout<<mat[i][j]<<" ";
        }
        else
        {
            for(int j=C-1;j>=0;j--)
                cout<<mat[i][j]<<" ";
        }
    }
}

int main()
{
    int arr[R][C]={{1,2,3,4},
                   {5,6,7,8},
                   {9,10,11,12},
                   {13,14,15,16}};

    printSnake(arr);

    return 0;
}
```

OUTPUT:

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

Matrix Boundary Traversal

```cpp
#include<iostream>
using namespace std;

const int R=4,C=4;

void bTraversal(int mat[R][C])
{
    if(R==1)
    {
        for(int i=0;i<C;i++)
            cout<<mat[0][i]<<" ";
    }
    else if(C==1)
    {
        for(int i=0;i<R;i++)
            cout<<mat[i][0]<<" ";
    }
    else
    {
        for(int i=0;i<C;i++)
            cout<<mat[0][i]<<" ";
        for(int i=1;i<R;i++)
            cout<<mat[i][C-1]<<" ";
        for(int i=C-2;i>=0;i--)
            cout<<mat[R-1][i]<<" ";
        for(int i=R-2;i>=1;i--)
            cout<<mat[i][0]<<" ";
    }
}

int main()
{
    int arr[R][C] = {{1, 2, 3, 4},
                     {5, 6, 7, 8},
                     {9, 10, 11, 12},
                     {13, 14, 15, 16}};

    bTraversal(arr);

    return 0;
}
```

OUTPUT:

1 2 3 4 8 12 16 15 14 13 9 5

Transpose of Matrix

Niave:

```
#include<iostream>
using namespace std;
const int n=4;
void transpose(int mat[n][n])
{
    int temp[n][n];

    for(int i=0;i<n;i++)
        for(int j=0;j<n;j++)
            temp[i][j]=mat[j][i];

    for(int i=0;i<n;i++)
        for(int j=0;j<n;j++)
            mat[i][j]=temp[i][j];
}
int main()
{
    int arr[n][n] = {{1, 2, 3, 4},
                     {5, 6, 7, 8},
                     {9, 10, 11, 12},
                     {13, 14, 15, 16}};

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

        cout << endl;
    }
    return 0;
}
```

OUTPUT:

1 5 9 13

2 6 10 14

3 7 11 15

4 8 12 16

Transpose of Matrix

Efficient :

```
#include<iostream>
using namespace std;

const int n=4;
 void transpose(int mat[n][n])
 {
     for(int i=0;i<n;i++)
        for(int j=i+1;j<n;j++)
            swap(mat[i][j],mat[j][i]);
 }

 int main()
{
    int arr[n][n] = {{1, 2, 3, 4},
                     {5, 6, 7, 8},
                     {9, 10, 11, 12},
                     {13, 14, 15, 16}};

   transpose(arr);

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

        cout << endl;
    }

    return 0;
}
```

OUTPUT:

1 5 9 13

2 6 10 14

3 7 11 15

4 8 12 16

Rotate matrix anti-clockwise by 90

Naïve:

```
#include<iostream>
using namespace std;
const int n=4;
void transpose(int mat[n][n])
{
    int temp[n][n];

    for(int i=0;i<n;i++)
        for(int j=0;j<n;j++)
            temp[n-j-1][i]=mat[i][j];

    for(int i=0;i<n;i++)
        for(int j=0;j<n;j++)
            mat[i][j]=temp[i][j];
}
int main()
{
    int arr[n][n] = {{1, 2, 3, 4},
                     {5, 6, 7, 8},
                     {9, 10, 11, 12},
                     {13, 14, 15, 16}};

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

        cout << endl;
    }
    return 0;
}
```

OUTPUT:

4 8 12 16

3 7 11 15

2 6 10 14

1 5 9 13

Rotate matrix anti-clockwise by 90

Efficient:

```
//1- find transpose of matrix
///2 reverse indivisual columns
#include<bits/stdc++.h>
using namespace std;
const int n=4;
void transpose(int mat[n][n])
{
    for(int i=0;i<n;i++)
        for(int j=i+1;j<n;j++)
            swap(mat[i][j],mat[j][i]);

    for(int i=0;i<n;i++)
    {
        int low=0,high=n-1;
        while (low<high)
        {
            swap(mat[low][i],mat[high][i]);
            low++;
            high--;
        }
    }
}

int main()
{
    int arr[n][n] = {{1, 2, 3, 4},
                     {5, 6, 7, 8},
                     {9, 10, 11, 12},
                     {13, 14, 15, 16}};

   transpose(arr);

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

        cout << endl;
    }

    return 0;
} OUTPUT same as previous problem
```

Spiral Traversal Matrix

```
#include<iostream>
using namespace std;
const int R=4,C=4;
void printSpiral(int mat[4][4], int R, int C)
{
    int top=0, left=0, bottom=R-1, right=C-1;
    while (top<=bottom && left<=right)
    {
        for(int i=left;i<=right;i++)
            cout<<mat[top][i]<<" ";

        top++;

        for(int i=top;i<=bottom;i++)
            cout<<mat[i][right]<<" ";

        right--;

        if(top<=bottom)
        {
            for(int i=right;i>=left;i--)
                cout<<mat[bottom][i]<<" ";

            bottom--;
        }

        if(left<=right)
        {
            for(int i=bottom; i>=top;i--)
                cout<<mat[i][left]<<" ";

            left++;
        }
    }
}

int main()
{
    int arr[R][C] = {{1, 2, 3, 4},
                     {5, 6, 7, 8},
                     {9, 10, 11, 12},
                     {13, 14, 15, 16}};

  printSpiral(arr, R, C);
    return 0;
} OUTPUT: 1 2 3 4 8 12 16 15 14 13 9 5 6 7 11 10
```

Search in Row wise and Column Wise sorted Matrix

Naïve:

```
#include<iostream>
using namespace std;

const int R=4, C=4;

void search(int mat[R][C], int x)
{
    for(int i=0;i<R;i++)
    {
        for(int j=0;j<C;j++)
        {
            if(mat[i][j]==x)
            {
                cout<<"Found at ("<<i<<", "<<j<<")";

                return;
            }
        }
    }

        cout<<"Not Found";
}

int main()
{
    int arr[][C] = {{10, 20, 30, 40},
                    {15, 25, 35, 45},
                    {27, 29, 35, 45},
                    {32, 33, 39, 50}};

    int x = 29;

    search(arr, x);

    return 0;
}
```

OUTPUT:

Found at (2, 1)

Search in Row wise and Column Wise sorted Matrix

Efficient:

```
#include<iostream>
using namespace std;
//time complexity o(R+C)
const int R=4,C=4;
void search(int mat[R][C], int x)
{
    int i=0,j=C-1;

    while(i<R && j>=0)
    {
        if(mat[i][j]==x)
        {
            cout<<"Found at ("<<i<<", "<<j<<")";
            return;
        }
        else if(mat[i][j]>x)
            j--;
        else
            i++;
    }

    cout<<"Not Found";
}

int main()
{
    int arr[][C] = {{10, 20, 30, 40},
                    {15, 25, 35, 45},
                    {27, 29, 35, 45},
                    {32, 33, 39, 50}};

    int x = 29;

    search(arr, x);

    return 0;
}
```

OUTPUT:

Found at (2, 1)

Median Of Row Wise Sorted Matrix

```cpp
// Iterator lower_bound (Iterator first, Iterator last, const val)
// Iterator upper_bound (Iterator first, Iterator last, const val)
// lower_bound returns an iterator pointing to the first element in
// the range [first,last) which has a value not less than ‘val’.

// and if the value is not present in the vector then it returns the end
iterator.
// upper_bound returns an iterator pointing to the first element in the
// range [first,last) which has a value greater than ‘val’.

#include<bits/stdc++.h>
using namespace std;

const int MAX=100;

int matMed(int mat[][MAX], int r, int c)
{
    int min=mat[0][0], max=mat[0][c-1];

    for(int i=1; i<r; i++)
    {
        if(mat[i][0]<min)
            min=mat[i][0];

        if(mat[i][c-1]>max)
            max=mat[i][c-1];
    }

    int medPos=(r*c+1)/2;

    while(min<max)
    {
        int mid=(min +max)/2;
        int midPos=0;

        for(int i=0;i<r;i++)
            midPos+=upper_bound(mat[i],mat[i]+c,mid)-mat[i];

        if(midPos<medPos)
            min=mid+1;

        else
            max=mid;

    }
```

```
    return min;
}

int main()
{
    int r=3, c=5;
    int m[][MAX]={{5,10,20,30,40},{1,2,3,4,6},{11,13,15,17,19}};

    cout<<"Median is "<<matMed(m,r,c)<<endl;
    return 0;
}
```

OUTPUT :

Median is 11