Page No.
Date

Q [leetcode 443.] String Compression

Eg: Input: chars = ["a", "a", "b", "b", "c", "c", "c"]
(a,a → 2; b,b → 2; c,c,c → 3)

Return 6

["a", "2", "b", "2", "c", "3"]

Eg: chars = ["a", "b", "b", "b", "b" ... "b"]
("a" → 1; b ... b → 12)

return 4

["a", "b", "1", "2"]

Code:

```
class Solution {
public:
    int compress(vector<char>& chars)
    {
        int i = 0;
        int ansIndex = 0;
        int n = chars.size();
```

Page No.
Date

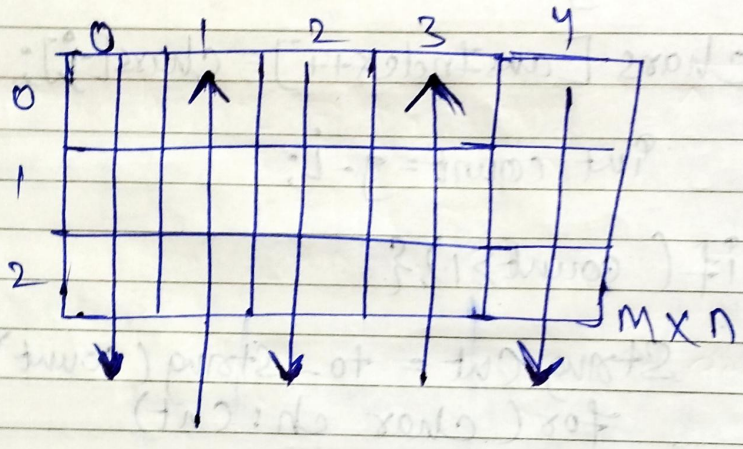
```
while(i<n){
    int j=i+1;
    while(j<n && chars[i]==chars[j])
        {
          j++;
        }
    chars[ansIndex++]=chars[i];
      int count=j-i;
    if(count>1){
      String cnt=to_String(count);
       for(char ch:cnt)
          { chars[ansIndex++]=ch;
          }
    }
      i=j;
  } return ansIndex;
}
};
```

Page No.
Date

Q. Why do we need to specify the column size, when passing a 2D array as a parameter?

Imp

Q. Print a vector of vector element in wave format.



Hint: Col Index → odd
Bottom to top.

Col Index → even
top to Bottom.

Code:
```
vector<int> wavePrint(vector<vector<int>>
                      int nRows, int mCols)
{
    vector<int> ans;
```

Page No.
Date

```
for (int col = 0; col < mCols; col++)
{
    // For odd columns.
    if (col & 1) {
        for (int row = nRows-1; row >= 0; row--)
        {
            ans.push_back (arr[row][col]);
        }
    else
        {
            // for even columns.
            for (int row=0; row<nRows; row++)
            {
                ans.push_back (arr[row][col]);
            }
        }
    }
    return ans;
}
```

int>> arr,

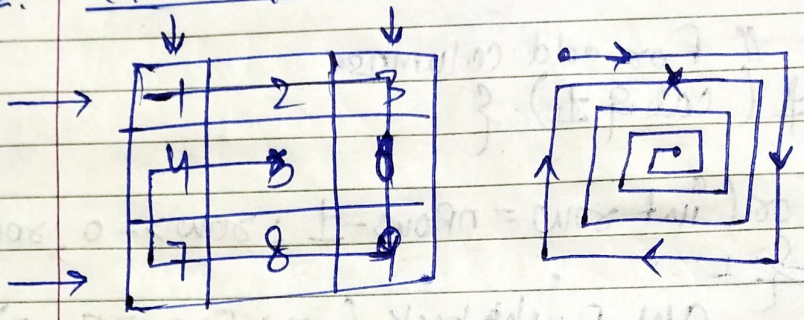
Time Complexity: O(m×n)

Page No.
Date

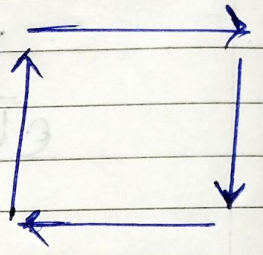
* leetcode 54. Spiral matrix

Imp Q.

Spiral print :



Approach : Starting row
ending column
ending row
Starting column.



and update after every step

Code : Class solution {

public:

```
vector<int> spiralorder(vector<vector<int>>& matrix)
{
    vector<int> ans;

    int row = matrix.size();
    int col = matrix[0].size();
```

Page No.
Date

```
int count = 0;
int total = row * col;

// Index initialization

int startingrow = 0;
int startingcol = 0;
int endingrow = row - 1;
int endingcol = col - 1;

while ( count < total){

  // printing starting row
for( int index = startingcol ; count<total &&
        index <= endingcol;  index ++)
   {
      ans.push_back( matrix[startingrow][index]);
        count ++;
   }
   startingrow ++;

  // printing endingcol
for( int index = startingrow; count < total &&
        index <= endingrow;  index ++)
    {
      ans.push_back( matrix[ index][endingcol]);
        count ++;
    }
    endingcol --;
```

Page No.
Date

```
// printing ending row
for( int index = ending col ; count < total &&
     index >= starting col ; index --)
  {
  ans.push_back ( matrix [ending row] [index]);
    count++;
  } ending row --;

// printing first col
for ( int index = ending row; count < total &&
     index >= starting row ; index --)
   {
  ans.push_back (matrix [index] [starting col]);
    count++;
  } starting col ++;

}  return ans;
  }
};
```

Page No.
Date

★ Q. Leetcode 74.
Search 2D matrix. (I)

1	3	5	7
10	11	16	20
23	30	34	60.

mxn

matrix = [[1,3,5,7], [10,11,16,20], [23,30,34,60]]

target = 3

o/p = true.

Approach: Same as linear binary search
Just . e = (row x col) - 1.

Code:

```
class Solution {
Public:
  bool searchmatrix(vector<vector<int>>&matrix
                    ,int target)
  {
      int row = matrix.size();
      int col = matrix[0].size();

      int start = 0;
      int end = (row * col) - 1;

      int mid = start + (end-start)/2;
```

Page No.
Date

```
while(start <= end) {
int element = matrix[mid/col][mid%col];
  if (element == target) {
        return 1;
  }
  if (element < target) {
        start = mid + 1;
  }
  else {
        end = mid - 1;
  }
  mid = start + (end - start)/2;
}
return 0;
}
};
```

Time complexity :- O(log(mn))

Page No.
Date

Leetcode 240.

Search a 2D matrix II

Eg:-

	0	1	2	3	4
0	1	4	7	11	15
1	2	5	8	12	19
2	3	6	9	16	22
3	10	13	14	17	24
4	18	21	23	26	30

5x5 (mxn)

Sorted in Row wise + Column wise

Approach :

Case 1 : if (element == target)
return 1

case 2 : if (element < target)
row ++;

Case 3 : if (element > target)
col--;

Hint : Start with the last column and first row

* Can't use mid method. due to sorted.

Page No.
Date

Code: Class Solution {

public:

bool SearchMatrix (vector<vector<int>>& matrix,
int target)

{

int row = matrix.size();
int col = matrix[0].size();

int rowindex = 0;
int colindex = col-1;

while (rowindex < row && colindex >=0)

{

int element = matrix[rowindex][colindex];

if (element == target) {
return 1;
}
if (element < target) {
rowindex ++;
}
else {
colindex --;
}

} return 0;

};

→ To escape the TLE in prime no.

Page No.
Date

(#) Sieve of Eratosthenes : Algorithm to check for prime numbers under n numbers.

Eg:- n = 40

~~1~~ (2) (3) ~~4~~ (5) ~~6~~ (7) ~~8~~ ~~9~~
~~10~~ (11) ~~12~~ (13) ~~14~~ ~~15~~ ~~16~~ (17)
~~18~~ (19) ~~20~~ ~~21~~ ~~22~~ (23) ~~24~~
~~25~~ ~~26~~ ~~27~~ ~~28~~ (29) ~~30~~
(31) ~~32~~ ~~33~~ ~~34~~ ~~35~~ ~~36~~ (37)
~~38~~ ~~39~~ ~~40~~

∴ Always start with i = 2 and till (n-1)

(1) Let us assume that every number is prime no.
(2) Now take 2 as prime ; now cancel out all the multiple of 2.
(3) Now take 3 as prime ; again cancel all multiple of 3.
(4) Repeat this process again & again.
(5) update count after every prime no.

Page No.
Date

Q 204. Leetcode (Count prime)

Given an integer n, return the number of prime numbers that are strictly less than n.

Code :

```
Class Solution {
private:
        bool isprime(int n) {
            if(n<=1){
                return false;
            }
            for(int i=2; i<n; i++)
            {if( n%i==0)
                {return false;
                }
            }
            return true;
        }
```

Page No.
Date

```
public:
    int countPrimes(int n){
        int cnt=0;
    for(int i=2; i<n; i++)
        {
            if(isprime(i)){
                cnt++;
            }
        }
        return cnt;
    }
};
```

→ logically correct ✓

Error: Time limit exceeded (TLE)

Time complexity = $O(n^2)$.

Explain:

2 → 2 case
3 → 3 case
4 → 4 case
⋮
$(n-1)^{th}$ term → $(n-1)$ case
n^{th} term → n case

$n(n-1) = n^2 - n$
$= n^2$

Page No.
Date

Code :

```
Class Solution {
public:
    int countprime( int n) {
        int cnt= 0;
        vector <bool> prime( n+1 , true);
        prime [0] = prime [1] = false;
        for( int i=2; i< n; i++){
            if( prime[i]){
                cnt++;
                for( int j= 2*i ; j<n; j=j+i){
                    Prime[j] = 0;
                }
            }
        }
    }
    return cnt;
}
};
```

Explanation
Cutting out the multiple of non prime for next no. step.

T.C of prime no.

Time complexity = O(n* log(log n))

Page No.
Date

Q. Segmented Sieve ? Code ?

GCD / HCF by ~~Eucledian~~ Euclidean Algorithm.

* $gcd(a,b) = gcd(a-b, b)$
$= gcd(a\%b, b)$

The Euclidean Algo for GCD(A,B) is

(a) if A=0 , GCD(A,B) = B ∴ GCD(0,B) = B
(b) if B=0 , GCD(A,B) = A ∴ GCD(A,0) = A
(c) if A≠0 , B≠0 use long form.

Eg:- $gcd(72,24) = gcd(a-b, b)$

$= gcd(72-24, 24)$
$= gcd(48, 24)$
$= gcd(24, 24)$
$= gcd(0, 24)$
$= 24$ ✓

* $lcm(a,b) * gcd(a,b) = a*b$

Page No.
Date

Code :

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

int gcd (int a, int b){

    if (a==0)
     return b;

    if (b==0)
     return a;

    while( a!=b){

      if (a>b)
         { a= a-b;
         }
      else {
            b= b-a;
          }
    } return a;
}
int main()
{  int a,b;
   cout<<" Enter the value of a&b";
    cin>> a>>b;

   int ans = gcd(a,b);
cout<<" Answer of gcd(a,b) is "<< ans;
 cout<<endl;    }
```

Page No.
Date

(*) Modulo operator properties :→

$(a+b)\%m = a\%m + b\%m$

$a\%m - b\%m = (a-b)\%m$

$a\%m * b\%m = (a \times b)\%m$

Q. fast power / exponentation ?

Q. Pigeonhole principle ?

Q. Catalon number ?

Q. Inclusion & Exclusion principle ?

Q. find factonal of 212 !

$212 \% m$ using the Arthimetic modulor.
$m = 10^9 + 7$.

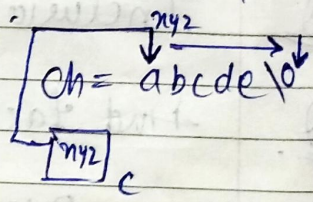
Page No.
Date

(*) Character Pointer :

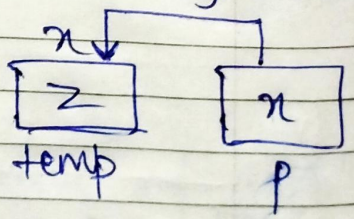
Eg:-
```
#include <iostream>
using namespace std;
int main() {
    int arr[5] = {1,2,3,4,5}
    char ch[6] = "abcde";
```

(String (c-style))

```
    cout << arr << endl;
// address of arr

    cout << ch << endl;
// abcde ∵ the implementation of cout for ch is different.
```




```
    char *c = &ch[0];

    cout << c << endl;
// abcde ∵ again the c pointing to ch and will print whole string where c is pointed. until '\0'.
```




```
    char temp = 'z';
    char *p = &temp;

    cout << p << endl;
// z???...] ∵ till it doesn't get '\0' null character.
}
```

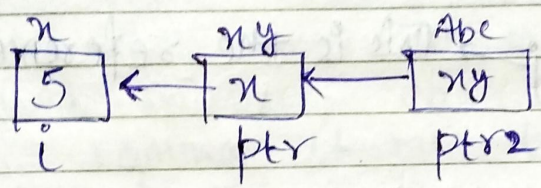
Page No.
Date

(*) Double ptr pointer :-

```
int i=5;
int *ptr = &i;
int **ptr2= &ptr;
```

→ double ptr

↓ pointer of pointer



Symbol table
i → x
ptr → xy
ptr2 → Abc

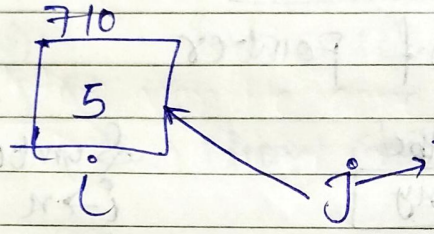
```
cout<< i<< endl;      // 5
cout<< *ptr<<endl;    // 5
cout<< **ptr2 <<endl; // 5.
cout<< &i<< endl;     // x
cout<< ptr<<endl;     // x
cout<< *ptr2 <<endl;  // x
```

Page No.
Date

(*) **Reference variable :-** // Pass by reference.

Same memory but different name.

Eg:- int i = 5;
int &j = i;



→ Reference variable is used as call by reference. Send the address to another variable to make the changes in it.

Eg:-

```
void update(int &n){
    n++;
}
```

Reference variable.

```
int main(){
    int n=5;
    cout<<"Before n is" <<n <<endl;//5
    update(n);
    cout<<"After n is" <<n <<endl;//6
```

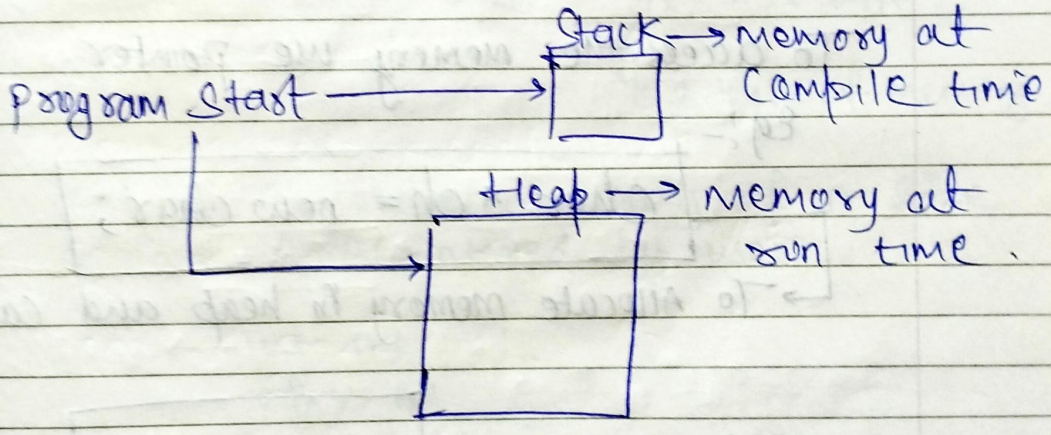
Page No.
Date

Heap & Stack Memory :-

(*) int n;
cout<< "Enter n" <<endl;
cin>>n;

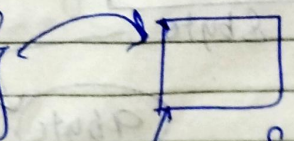
int arr[n]; → BAD practise ✗
↳ because our program can crash due to lack of stack memory.

We must know the size of array during compile time like arr[1000];
But In above case, the size of arr is coming at run time which is bad practise.



(*) The variables we declare in our program are used the stack memory

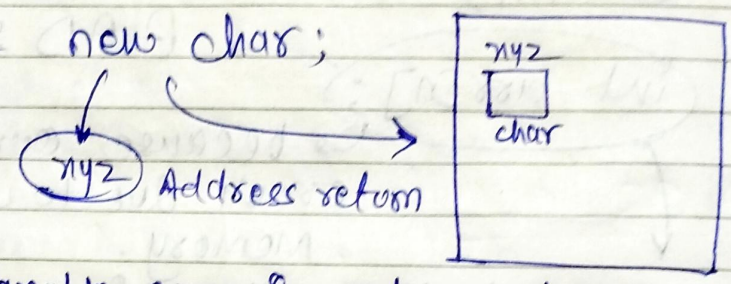
Eg:- int n=5;
char ch;
int arr[50];



Page No.
Date

(*) If we want to use the Heap memory we have to use the new keyword.

Eg:- new char;



Note: variable name is not allowed in Heap → Heap { Dynamic allocation }

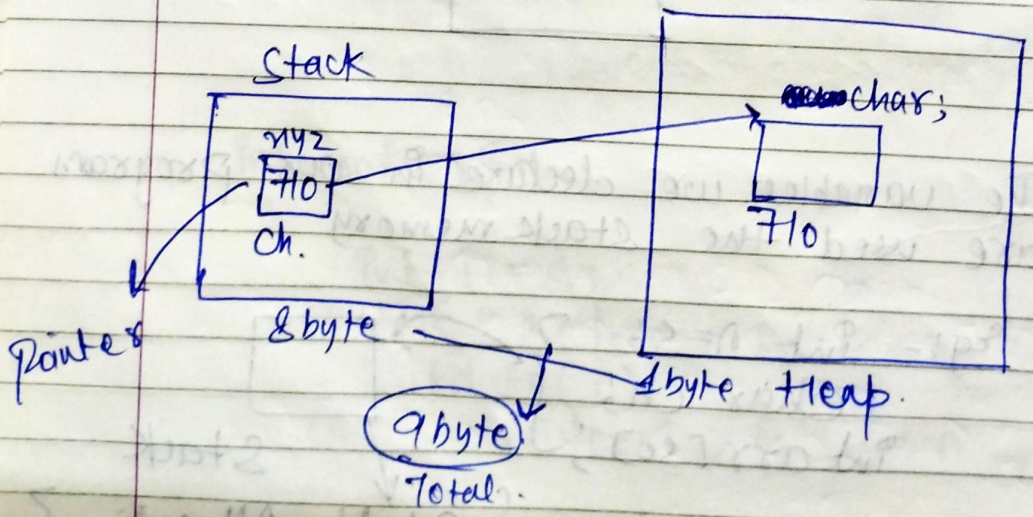
Note: Heap memory return the address.

To access the memory use pointer

Eg:-

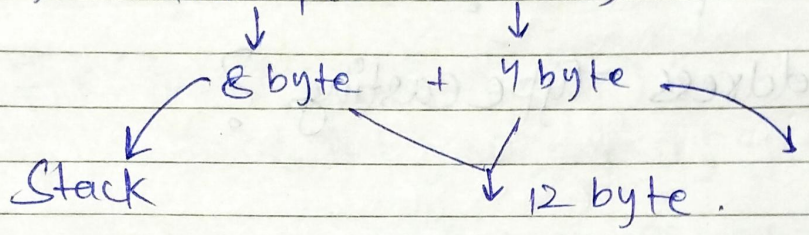
```
char *ch = new char;
```

→ To Allocate memory in heap and can use it

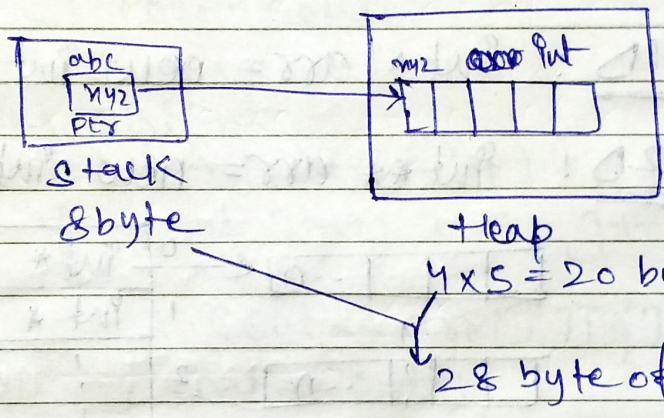


Page No.
Date

Eg:- `int *ptr = new int;`



Eg:- `int *ptr = new int[5];`



Eg:-
```
// To create an array of n size
   n is Entered by user

int n;
cin >> n;

int *arr = new int[n];
```

(*) To release the memory use 'delete' keyword.

Eg:- `delete arr;`

Page No.
Date

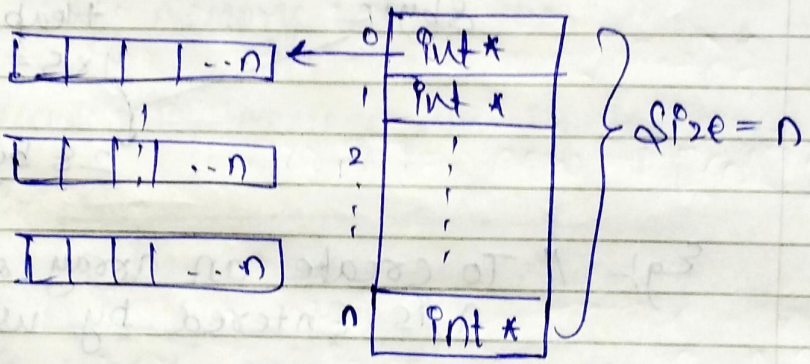
Q Void pointer ?

Q Address type casting ?

Dynamic memory Allocation of 2D Array.

1D : int* arr = new int[n];

2D : int** arr = new int*[n];



→ n×n 2D array.

Eg:-
```
int main()
{
    int n;
    cin>>n;

    // Creating a 2D array
    int** arr = new int*[n];
    for(int i=0; i<n; i++) {
        arr[i] = new int[n]; }
```

Page No.
Date

```
// taking input
for(int i=0; i<n; i++) {
    for(int j=0; j<n; j++){
        cin>>arr[i][j];
    }
} cout<<endl;

// taking output
for(int i=0; i<n; i++) {
    for(int j=0; j<n; j++){
        cout<<arr[i][j];
    } cout<<endl;
}
```

Result:

n = (3)

1 2 3 4 5 6 7 8 9

1 2 3
4 5 6
7 8 9

nxn

row ↑ col ↑

Page No.
Date

Eg:- m x n 2D dynamic array (Heap)

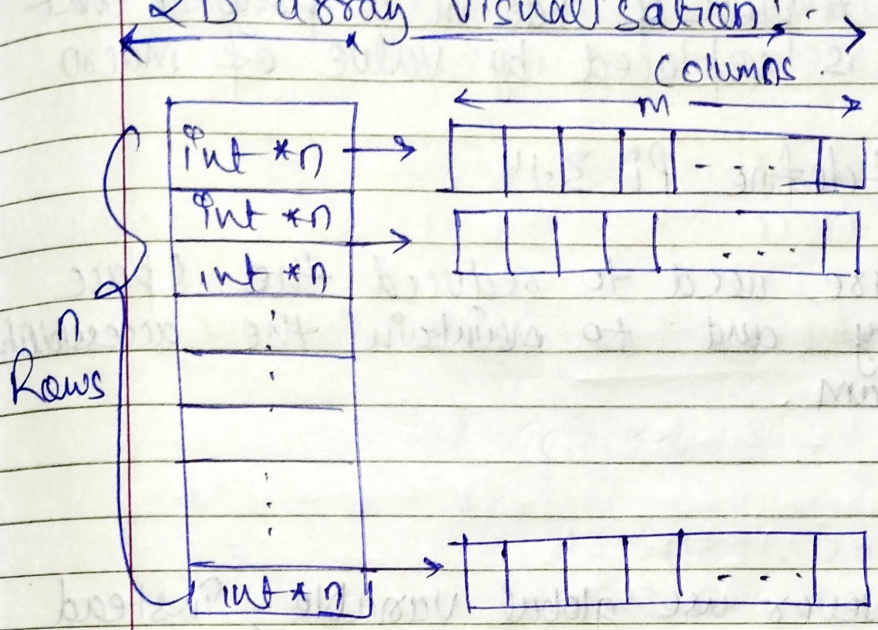
```
int main(){

    int row;
    cin>> row;

    int col;
    cin>> col;

// Creating m x n 2D array.
    int** arr= new int*[row];

    for(int i=0; i<row; i++){

        arr[i]= new int[col];
    }

// taking input
    for(int i=0; i<row; i++){
        for(int j=0; j<col; j++){

            cin>> arr[i][j];
        }
    }

// taking output
    for(int i=0; i<row; i++){
        for(int j=0; j<col; j++){

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

n Rows

H/W Q

Page No.
Date

2D array visualisation:-



// Releasing memory for m columns.

```
for(int i=0; i<row; i++){
    delete [] arr[i];
}
```

// Releasing memory of n Rows.

```
delete [] arr;
```

H/W

Q Jagger array ? create it dynamically.