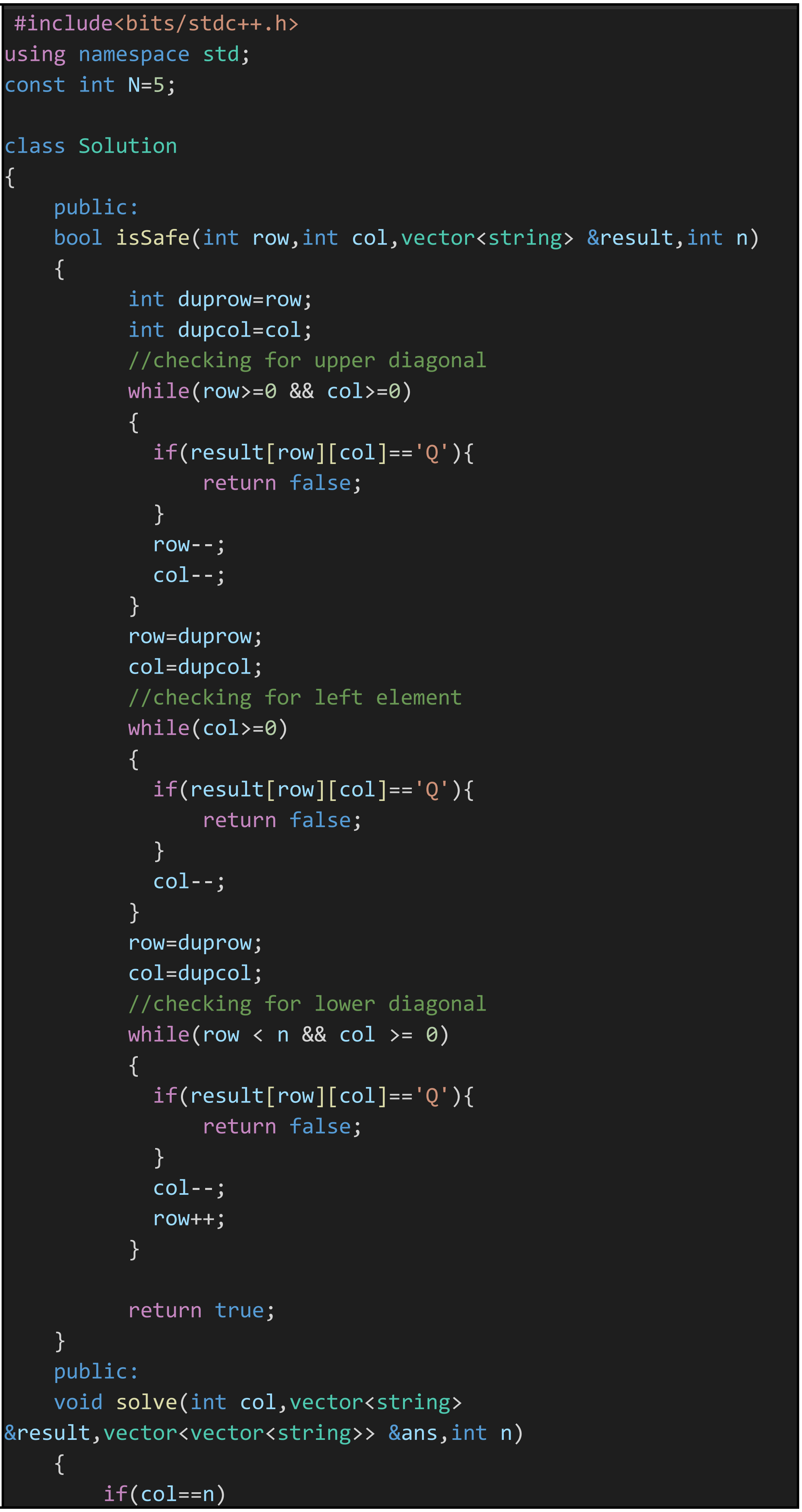
NAME:	Ameya Dabholkar
UID:	2021300023
SUBJECT	DAA
EXPERIMENT NO :	07
AIM:	To solve the N-Queen Problem
PROBLEM STATEMENT 1:	
THEORY	Now, we place queen q_1 in the very first acceptable position (1, 1). Next, we put queen q_2 so that both these queens do not attack each other. We find that if we place q_2 in column 1 and 2, then the dead end is encountered. Thus the first acceptable position for q_2 in column 3, i.e. (2, 3) but then no position is left for placing queen 'q_3' safely. So we backtrack one step and place the queen 'q_2' in (2, 4), the next best possible solution. Then we obtain the position for placing 'q_3' which is (3, 2). But later this position also leads to a dead end, and no place is found where 'q_4' can be placed safely. Then we have to backtrack till 'q_1' and place it to (1, 2) and then all other queens are placed safely by moving q_2 to (2, 4), q_3 to (3, 1) and q_4 to (4, 3). That is, we get the solution (2, 4, 1, 3). This is one possible solution for the 4-queens problem. For another possible solution, the whole method is repeated for all partial solutions. The other solutions for 4 - queens problems is (3, 1, 4, 2) i.e.

	1	2	3	4
1			q_1	
2	q_2			
3				q_3
4		q_4		

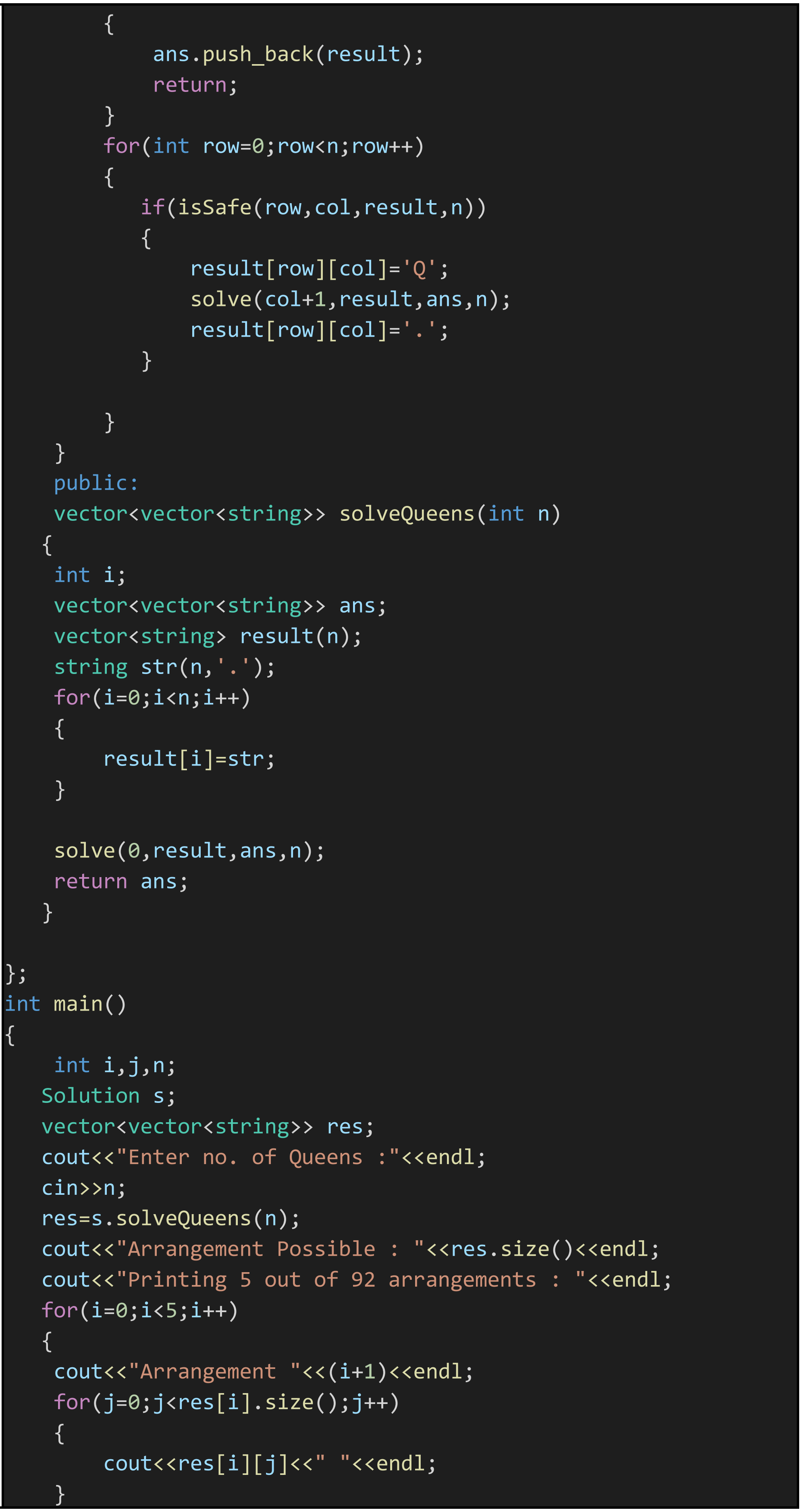
ALGORITHM

```
N - Queens (k, n)
{
  For i ← 1 to n
      do if Place (k, i) then
  {
    x [k] ← i;
    if (k ==n) then
      write (x [1....n));
    else
    N - Queens (k + 1, n);
  }
}
```

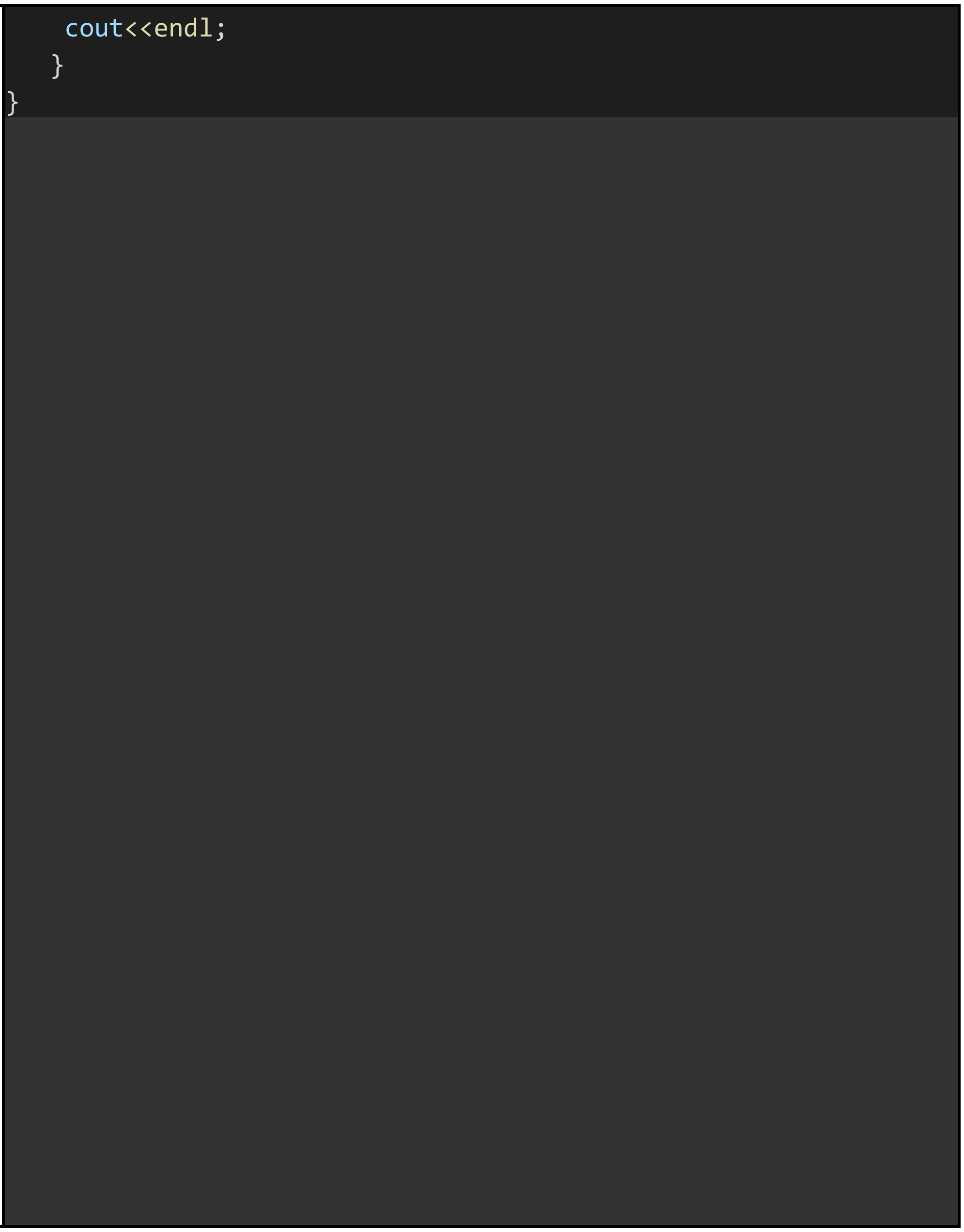
PROGRAM:



```
 #include<bits/stdc++.h>
using namespace std;
const int N=5;

class Solution
{
    public:
    bool isSafe(int row,int col,vector<string> &result,int n)
    {
          int duprow=row;
          int dupcol=col;
          //checking for upper diagonal
          while(row>=0 && col>=0)
          {
            if(result[row][col]=='Q'){
                return false;
            }
            row--;
            col--;
          }
          row=duprow;
          col=dupcol;
          //checking for left element
          while(col>=0)
          {
            if(result[row][col]=='Q'){
                return false;
            }
            col--;
          }
          row=duprow;
          col=dupcol;
          //checking for lower diagonal
          while(row < n && col >= 0)
          {
            if(result[row][col]=='Q'){
                return false;
            }
            col--;
            row++;
          }

          return true;
    }
    public:
    void solve(int col,vector<string>
&result,vector<vector<string>> &ans,int n)
    {
        if(col==n)
```

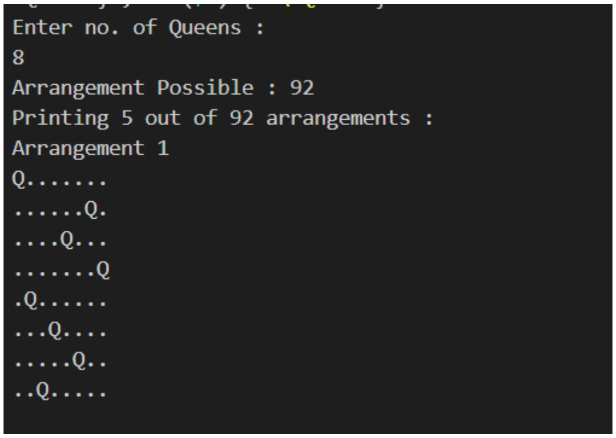


```
        {
            ans.push_back(result);
            return;
        }
        for(int row=0;row<n;row++)
        {
           if(isSafe(row,col,result,n))
           {
               result[row][col]='Q';
               solve(col+1,result,ans,n);
               result[row][col]='.';
           }

        }
    }
    public:
    vector<vector<string>> solveQueens(int n)
   {
    int i;
    vector<vector<string>> ans;
    vector<string> result(n);
    string str(n,'.');
    for(i=0;i<n;i++)
    {
        result[i]=str;
    }

    solve(0,result,ans,n);
    return ans;
   }

};
int main()
{
    int i,j,n;
   Solution s;
   vector<vector<string>> res;
   cout<<"Enter no. of Queens :"<<endl;
   cin>>n;
   res=s.solveQueens(n);
   cout<<"Arrangement Possible : "<<res.size()<<endl;
   cout<<"Printing 5 out of 92 arrangements : "<<endl;
   for(i=0;i<5;i++)
   {
    cout<<"Arrangement "<<(i+1)<<endl;
    for(j=0;j<res[i].size();j++)
    {
        cout<<res[i][j]<<" "<<endl;
    }
```

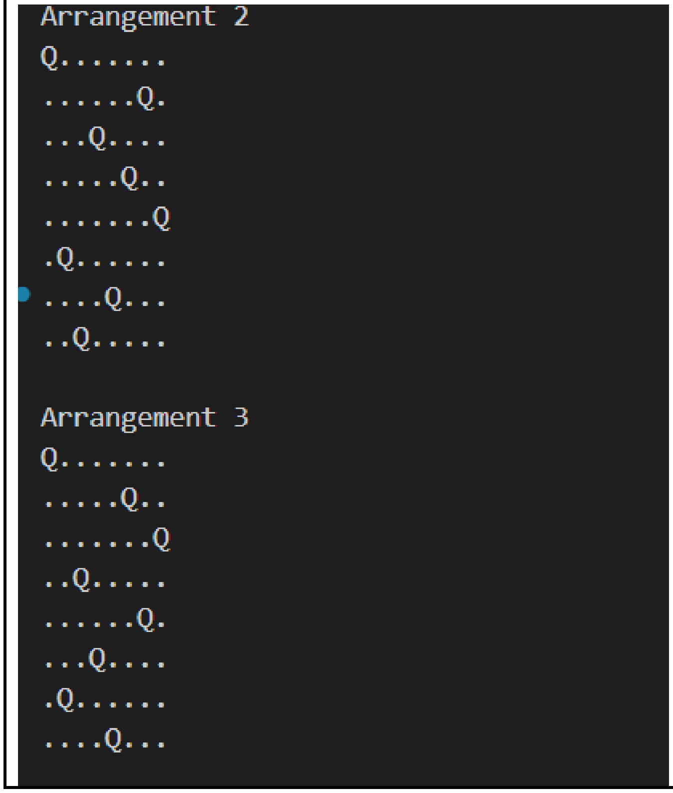


```
    cout<<endl;
   }
}
```

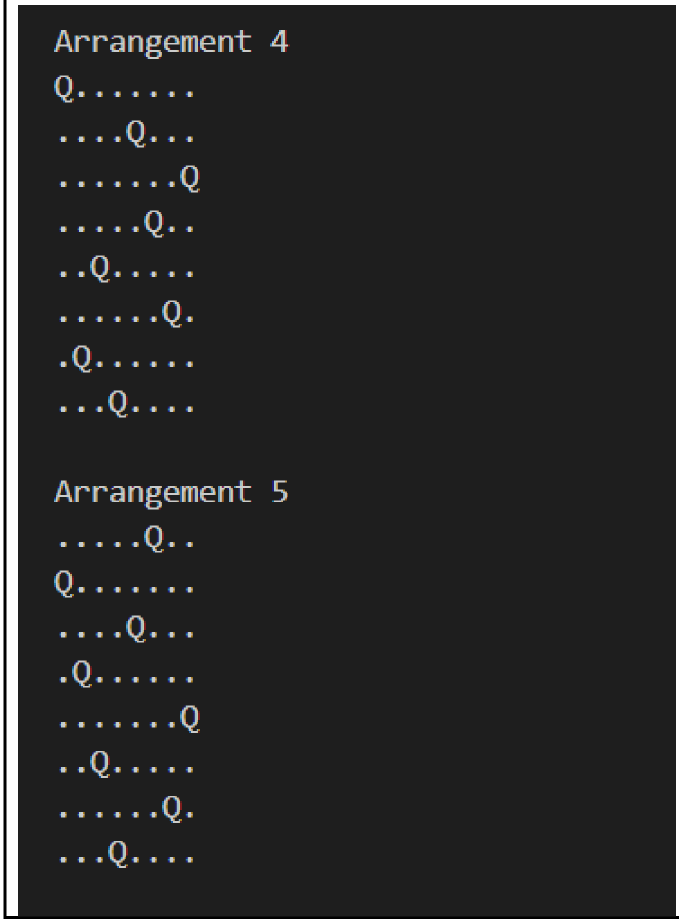
RESULT (SNAPSHOT)



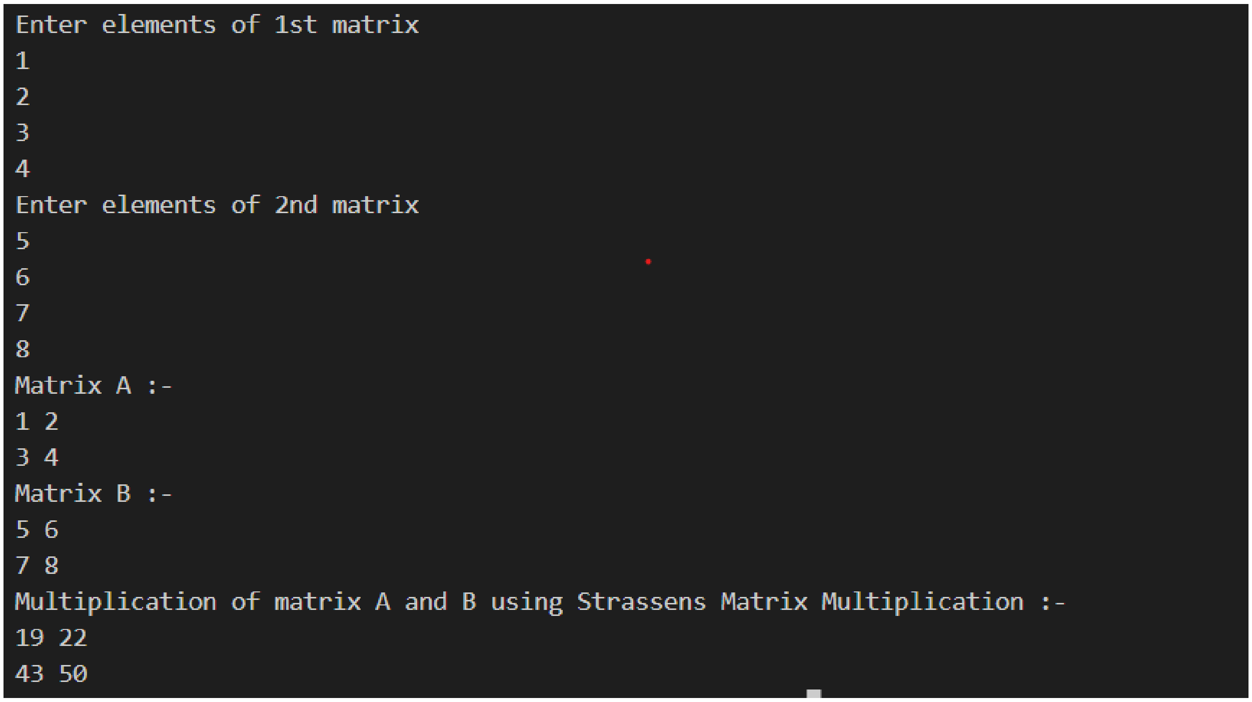
```
Enter no. of Queens :
8
Arrangement Possible : 92
Printing 5 out of 92 arrangements :
Arrangement 1
Q.......
......Q.
....Q...
.......Q
.Q......
...Q....
.....Q..
..Q.....
```



```
Arrangement 2
Q.......
......Q.
...Q....
.....Q..
.......Q
.Q......
....Q...
..Q.....

Arrangement 3
Q.......
.....Q..
.......Q
..Q.....
......Q.
...Q....
.Q......
....Q...
```



```
Arrangement 4
Q.......
....Q...
.......Q
.....Q..
..Q.....
......Q.
.Q......
...Q....

Arrangement 5
.....Q..
Q.......
....Q...
.Q......
.......Q
..Q.....
......Q.
...Q....
```



```
Enter elements of 1st matrix
1
2
3
4
Enter elements of 2nd matrix
5
6
7
8
Matrix A :-
1 2
3 4
Matrix B :-
5 6
7 8
Multiplication of matrix A and B using Strassens Matrix Multiplication :-
19 22
43 50
```

CONCLUSION:	Through this experiment I understood how to implement strassens matrix multiplication