```
/*
 * Problem Statement :- Design n-Queens matrix having first Queen placed. Use
backtracking
 *                                  to place remaining Queens to generate the final n-
queen's matrix.
 *
 * Time Complexity  : O(n!)
 * Space Complexity : O(n)
 */

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

vector<vector<int>> grid;
vector<bool> col;
vector<bool> lrdiag;
vector<bool> rldiag;
int cnt = 0;

void display()
{
	for(auto itr:grid)
	{
		cout<<"\t ";
		for(auto x:itr)
		{
			if(x)
			{
				cout<<"Q ";
			}
			else
			{
				cout<<"X ";
			}
		}
		cout<<endl;
	}
	cout<<endl;
}

bool is_safe(int r, int c, int n)
{
	if(lrdiag[r-c+n-1] || rldiag[r+c] || col[c]) return false;
	return true;
}

void n_queen(int row, int n)
{
	if(row>=n)
	{
		display();
		cnt++;
//		exit(0);
		return;
	}
```

```
	for(int c=0; c<n; c++)
	{
		if(is_safe(row, c, n))
		{
			grid[row][c] = 1;
			col[c] = true;
			lrdiag[row-c+n-1] = true;
			rldiag[row+c] = true;

			n_queen(row+1, n);

			grid[row][c] = 0;
			col[c] = false;
			lrdiag[row-c+n-1] = false;
			rldiag[row+c] = false;
		}
	}
	return;
}

int main()
{
	int n, c;

	cout<<"\n\t Enter size of board : ";
	cin>>n;
	grid.assign(n, vector<int>(n,0));
	col.assign(n, false);
	lrdiag.assign(2*n-1, false);
	rldiag.assign(2*n-1, false);

	cout<<"\n\t Enter the column number where the first queen is placed : ";
	cin>>c;
	c--;			//0-based indexing
	grid[0][c] = 1;
	col[c] = true;
	lrdiag[n-1-c] = true;
	rldiag[0+c] = true;

	n_queen(1, n);
	if(cnt == 0)
	{
		cout<<"\n\t No Solution Exist !!"<<endl;
	}
	else
	{
		cout<<"\n\t Total possible solutions : "<<cnt<<endl;
	}
}
```