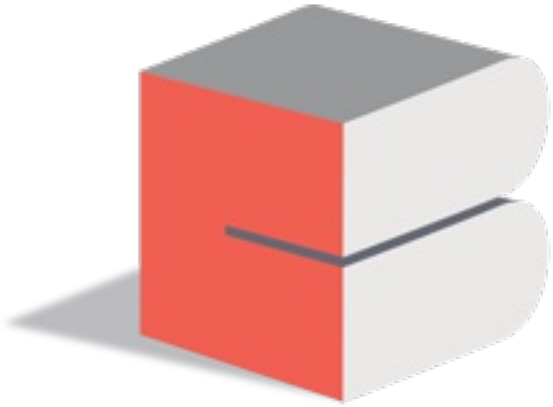

N-Queen Code - 3 Approaches

N-Queen Bitmasking Approach (More efficient)

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

int n;
int ans =0, DONE;
///rowmask denote which positions(colms) in all rows are fill
ed
/// ld, rd denotes unsafe positions along diagonals for the c
urrent row
/// done is vector of all 11111 ( n times 1 )
/// safe denotes the cols which are safe in the current row

/// Most optimisized n queen code !
void solve(int rowMask,int ld,int rd){

```

```
    if(rowMask==DONE){ ans++; return; }

    int safe = DONE&(~(rowMask|ld|rd));
    while(safe){
        int p = safe &(-safe);
        safe = safe - p;
        solve(rowMask|p, (ld|p)<<1, (rd|p)>>1);
    }
}


int main()
{
    cin>>n;
    DONE = ((1<<n) - 1);
    solve(0,0,0);
    cout<<ans<<endl;

}
```

N-Queen Bitset Approach

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

```cpp
bitset<30> col,d1,d2;

void solve(int r,int n,int &ans){
    if(r==n){ ans++; return;}

    for(int c=0;c<n;c++){
        //Safe Checking O(1)
        if( !col[c] && !d1[r-c+n-1] && !d2[r+c]){
            col[c] = d1[r-c+n-1] = d2[r+c] = 1;
            solve(r+1,n,ans);
            col[c] = d1[r-c+n-1] = d2[r+c] = 0;
        }
    }
}

int main(){
    int n;
    cin>>n;

    int ans = 0;
    solve(0,n,ans);
    cout<<ans<<endl;
return 0;
}
```

N-Queen Naive Backtracking Approach

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

bool isSafe(int board[][100],int i,int col,int n){
    `//Iteratively check over column, diagonals
    // O(n)

}

//Count and Print All Config
bool solveNQueen(int board[][100],int i,int n,int &cnt){
    //Base Case
    if(i==n){
        //Print the board
        cnt+=1;
        return true;
    }
    //Rec Case
    //Place the queen in the ith row
    int cnt = 0;
    for(int col=0;col<n;col++){
        //Check if i,col is safe to place the current queen
        if(isSafe(board,i,col,n)){
            //Place the queen
            board[i][col] = 1;
            //Solve the remaining subproblem
```

```
            bool nqueenSolved = solveNQueen(board,i+1,n);

            //Backtracking
            board[i][col] = 0;

        }
    }
    return false;
}

int main(){
    int board[100][100];
    int n;
    cin>>n;



}
```