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

#include "tmatrix.h"
#include "prompt.h"

class Queens
{
  public:
    Queens(int size);

    bool Solve();

    void Print(ostream& out) const;
  private:

    bool NoQueensAttackingAt(int r, int c) const;
    bool SolveAtCol(int col);
    tmatrix<bool> myBoard;
};

bool Queens::NoQueensAttackingAt(int r, int c) const
// pre: column c is clear
// post: return true if row clear and diagonals crossing at
//       (r,c) clear
{
    int k, row,col;

    // check row r
    for(k=0; k < myBoard.numcols(); k++) {
        if (k != c && myBoard[r][k]) return false;
    }

    // check diagonal northwest
    row = r-1; col = c-1;
    while (0 <= row && 0 <= col) {
        if (myBoard[row][col]) return false;
        row--; col--;
    }

    // check diagonal southeast
    row = r+1; col = c+1;
    while (row < myBoard.numrows() && col < myBoard.numcols()) {
        if (myBoard[row][col]) return false;
        row++; col++;
    }

    // check diagonal northeast
    row = r-1; col = c+1;
    while (0 <= row && col < myBoard.numcols()) {
        if (myBoard[row][col]) return false;
        row--; col++;
    }

    // check diagonal southwest
    row = r+1; col = c-1;
    while (row < myBoard.numrows() && 0 <= col) {
        if (myBoard[row][col]) return false;
        row++; col--;
    }

    return true;
}

Queens::Queens(int size)
  : myBoard(size,size,false)
// post: board initialized to hold no queens
{ }
```



```
bool Queens::Solve()
// post: returns true if n queens can be placed, false otherwise
{
    return SolveAtCol(0);
}

bool Queens::SolveAtCol(int col)
// pre: queens placed at columns 0,1,...,col-1
// post: returns true if queen can be placed in column col
//       if col == size of board, then n queens are placed
{
    int k;
    int rows = myBoard.numrows();
    if (col == rows) return true;

    for(k=0; k < rows; k++) {
        if (NoQueensAttackingAt(k,col)) {  // row k is clear, try it
            myBoard[k][col] = true;        // place the queen
            if (SolveAtCol(col+1)) {       // try remaining columns
                return true;               // placed them all!
            }
            myBoard[k][col] = false;       // couldn't place, remove queen
        }
    }
    return false;
}

void Queens::Print(ostream& out) const
// post: board printed, 'X' for queen
{
    int j,k;
    for(j=0; j < myBoard.numrows(); j++) {
        for(k=0; k < myBoard.numcols(); k++) {
            out << (myBoard[j][k] ? 'X' : '.');
        }
        out << endl;
    }
}

int main()
{
    int size = PromptRange("size of board: ",2,100);

    Queens nq(size);

    if (nq.Solve()) {
        nq.Print(cout);
    }
    else {
        cout << "no solution found" << endl;
    }
    return 0;
}
```