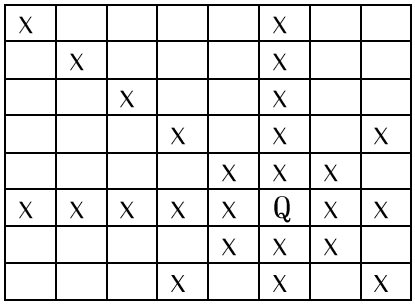
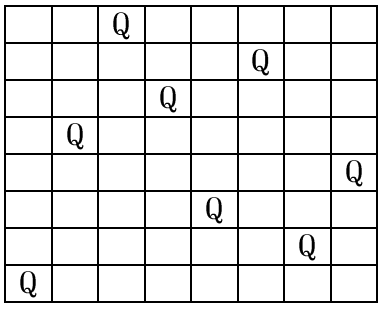
CSCI 689 **Computer Assignment 11** **Spring 2016**
(30 points)

N Queens Puzzle

The most powerful piece in the game of chess is the queen, which can move any number of squares in any direction, horizontally, vertically, or diagonally. For example, the queen shown in the following chessboard of size N = 8 can move to any of the marked squares.



Even though the queen can cover a large number of squares, it is possible to place N queens on an N x N chessboard so that none of them can attack any of the others, as shown in the following diagram:



Write a C++ program that solves the problem of whether it is possible to place N queens for N = 1, 2, ..., 8 on an N x N chessboard so that none of them can move to a square occupied by any of the others in a single turn. Your program should either display a solution if it finds one or report that no solutions exists.

The main ( ) routine simply reads several values of N, between 1 and 8, from the stdin, and for each value of N, it calls the following routine to place the N queens on the chessboard.

- void solveNQueens (const unsigned& n): This routine first creates a two-dimensional vector of vectors of type bool for the given size n. Then, it calls the routine initBoard ( ), to initialize a RNG with the seed value time (0) and set all positions on the chessboard to false. To place the n queens on the chessboard, starting from the first row, it prints out the chessboard size and calls to routine solveNQueensUtil ( ), as explained below. If the returned value of solveNQueensUtil ( ) is true, then it calls the routine printBoard ( ) to print out the contents of the chessboard on stdout by

showing the positions of the queens on the chessboard, otherwise, it prints a message indicating that a solution does not exists.

- bool solveNqueensUtil (vector < vector < bool > >& board, const unsigned& row): This *recursive* routine starts on the row number row and gets a random column number col, between 0 and (board.size ( ) − 1) from the RNG, and it checks if a queen can be placed on the location (row, col). It calls the routine isSafe ( ), as explained below, to determine if the location is safe, so the queen can be placed in that location. If the row is not the last row on the board, it continues to the next row by making a recursive call. If a queen cannot be placed on the column col, then it chooses another column in the given row, and if none of the columns in the row turns out invalid, then the returned value of the recursive call will be false. In this case, by *backtracking,* this routine simply returns to the previous row and chooses another column to replace the queen in that row. If the *backtracking* goes all the way to the first raw and none of the columns in that row results a successful placement, then the routine returns false to calling routine.

- bool isSafe (const vector < vector < bool > >& board, const int& row, const int& col): This routine checks if a queen can be placed in the row number row and the column number col on the board. If the answer is yes, then it returns true. If there is another queen in the location (i, j), then the row cannot be equal to i, since the queens are placed on the board one piece at a time, but those two queens can be in the same column if col == j, and it can be easily verified that they can be in the same diagonal if abs (row − i) == abs (col − j).

Put the declarations of all constants and routines that you use in your program in the header file prog11.h, and the implementation of all routines in the source file prog11.cc. At the top of your source file, insert the following statement: #include “prog11.h”.

To compile the source files and link the generated object files with the system library routines, first make a link to makefile, in directory: ~cs689/progs/16s/p11, from your working directory, and then execute: make N=11.

To test of your program, execute: make execute N=11. The correct output file, prog11.out, is in the same directory with makefile. Since you execute your program with a random seed value, you’ll get a different placement of the queens on the chessboard each time. For your information, the total number of possible placements is given as: 1 for N = 1; 0 for N = 2, 3; 2 for N = 4; 10 for N = 5; 4 for N = 6; 40 for N = 7; and 92 for N = 8, but some solutions differ only by *symmetry operations* (rotations and reflections) of the chessboard.

When your program is ready, mail its source and header files to your TA by executing: mail_prog prog11.cc prog11.h.