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.

x					x		
	x				x		
		x			x		
			x		x		x
				x	x	x	
x	x	x	x	x	Q	x	x
				x	x	x	
			x		x		x

Even though the queen can cover many 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:

		Q					
					Q		
			Q				
	Q						
							Q
				Q			
						Q	
Q							

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 calls the following routine for each value of N, between 1 and 8, 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 size n. Then, it calls the routine initBoard (), to initialize the RNG with the seed value of 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 the 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 location (row, col). It calls the routine isSafe (), which is 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 the 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 to be valid, 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 row and none of the columns in that row results a successful placement, then the routine returns false to the calling routine.

- **bool isSafe (const vector < vector < bool > >& board, const int& row, const int& col):** 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 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 prog8.h, and the implementation of all routines in the source file prog8.cc. At the top of your source file, insert the following statement: #include "prog8.h".