# IE523: Financial Computing
# Fall, 2019
## Programming Assignment 1: N Queens Problem
## Due Date: 6 September, 2019



The original version of this problem goes like this – You have a $8 \times 8$ chessboard, and you have to place 8 queens on this chessboard such that no two of them threaten each other. Since a queen can attack any piece that shares a row, column, or diagonal, with it, we are essentially looking to place 8 elements in a $8 \times 8$ grid such that no two of them share a common row, column, or diagonal. You can read more about this problem at this link.

The $n \times n$ version of this problem asks you to place $n$ queens on a $n \times n$ chessboard. For the first part of this assignment we are seeking just one solution (among a set of many possible solutions) to this problem. You have to solve this by a recursive implementation of the *backtracking* search/algorithm, and it must be done in an "object-oriented" manner. For the second part, we seek all possible solutions to the problem, by modifying the code for the first part.

Assume you have gone through the necessary steps to define a $n \times n$ chessboard. You must do the following:

1. Write a function is_this_position_safe($i, j$), which returns "true" (resp. "false") if placing a queen in the $(i, j)$-th position in the $n \times n$ chessboard is safe (resp. unsafe).

2. Implement a recursive back-tracking search procedure solve($i$), as shown in figure 2, which returns "true" if there is a way to place a queen at the $i$-th column of the $n \times n$ chessboard, where $0 \leq i \leq n-1$ (notice: the indexing starts from 0 and ends with $n-1$). You call solve(0) to solve the puzzle.

# Finding one solution to the $N$-Queens Problem

Just to get us thinking in object-oriented terms, I want you to do the following:

1. Define a class called Board, it should have a private data member called size (which is $n$ in the $n \times n$ chessboard), and a integer-valued chessboard. If there is a queen at the $(i, j)$-th position of the $n \times n$ chessboard, then chessboard$[i][j] = 1$, otherwise, chessboard$[i][j] = 0$.

2. Keep in mind, the value of $n (=$ size$)$ is not known a priori – **it will be known at runtime when the user inputs it via the command-line** (you should pay attention to this when I go over it in class). One way is to accomplish this is to have a private data member declared as int **chessboard, and once the size of the chessboard is known, you can declare an array using a *pointer-to-pointers* approach. If you need help with this

```
Boolean Solve(column)
if column ≥ n then
    You have solved the puzzle.
else
    for 0 ≤ row ≤ n − 1 do
        if is_this_position_safe(row, column) then
            Place queen at (row, column)-position in the n × n chessboard.
            if Solve(column+1) then
                Return true.
            else
                Remove queen at (row, column)-position in the n × n chessboard.
                Placing it here was a bad idea.
            end if
        end if
    end for
end if
{ /* If we got here then all assignments of the queen in (column-th column
are invalid. So, we return false*/}.
Return false.
```

Figure 1: Pseudo-code for the recursive implementation of the exhaustive-search algorithm for the $(n \times n)$ Queens-puzzle. You solve the puzzle by calling **Solve**(0), assuming the indices are in the range $\{0, 1, \ldots, n-1\}$.

```
// N Queens Problem via (Backtracking, which is implemented by) Recursion
// Written by Prof. Sreenivas for IE523: Financial Computing

#include <iostream>
#include "N_queens.h"

int main (int argc, char * const argv[])
{
    Board x;

    int board_size;
    sscanf (argv[1], "%d", &board_size);

        x.nQueens(board_size);

    return 0;
}
```

Figure 2: f16_prog1_hint.cpp: C++ code to help you with Programming Assignment #1.

part, you might want to see the handout "Programming Assignment 5: Dynamic Arrays in C++" in the Bootcamp folder on Compass.

3. I also want you to write a member function that prints the (solved) chessboard. Although I do not want you to mimic my output exactly, something similar will be sufficient.

4. I have provided partial code samples for the *.cpp file in figure 2, and the *.h file in figure 3. Two sample outputs are shown in figure 4.

Here is what I want from you for the first part of the assignment

1. A well-commented C++ code that produces output that is similar to what is shown in figure 4.

You will submit this electronically to the TA before the due date.

# Finding all solutions to the $N$-Queens Problem

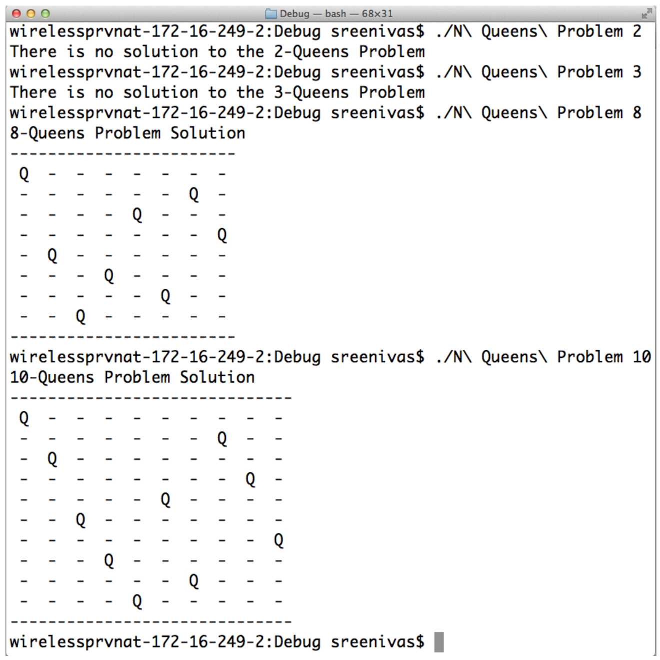
.

For this part of the assignment I want you to extend/modify the code for the previous part of the assignment, where you found a single solution to the $N$-Queens Problem, to find all solutions to the $N$-Queens problem.

Keep in mind that the number of solutions can grow to be very large very quickly. Table 1 shows the number of solutions for different values of $N$. I am looking for outputs along the lines of what is shown in figures 5, 6 and 7.

```cpp
#ifndef N_queens
#define N_queens
using namespace std;
class Board
{

        // private data member: size of the board
        int size;

    // pointer-to-pointer initialization of the board
        int **chess_board;

        // private member function:  returns 'false' if
        // the (row, col) position is not safe.
        bool is_this_position_safe(int row, int col)
        {
                // write the appropriate code on your own that returns
                // "true" if the (row,col) position is safe.  If it is
                // unsafe (i.e. some other queen can threaten this position)
                // return "false"

        }

        // private member function: initializes the (n x n) chessboard
        void initialize(int n)
        {
                size = n;

                // write the appropriate code that uses the pointer-to-pointer
                // method to initialize the (n x n) chessboard.  Once initialized,
                // put zeros in all entries.  Later on, if you placed a queen in
                // the (i,j)-th position, then chessboard[i][j] will be 1.
        }

        // private member function: prints the board position
        void print_board()
        {
        std::cout << size << "-Queens Problem Solution" << std::endl;
                // write the appropriate code here to print out the solved
                // board as shown in the assignment description

        }

        // private member function: recursive backtracking
        bool solve(int col)
        {
                // implement the recursive backtracking procedure described in
                // pseudocode format in figure 1 of the description of the first
                // programming assignment
        }

public:
        // Solves the n-Queens problem by (recursive) backtracking
        void nQueens(int n)
        {
                initialize(n);

                if (solve(0))
                        print_board();
                else
                        std::cout << "There is no solution to the " << n << "-Queens Problem" << std::endl;
        }
};
#endif
```

Figure 3: f16_prog1_hint.h: C++ code to help you with Programming Assignment #1.



```
wirelessprvnat-172-16-249-2:Debug sreenivas$ ./N\ Queens\ Problem 2
There is no solution to the 2-Queens Problem
wirelessprvnat-172-16-249-2:Debug sreenivas$ ./N\ Queens\ Problem 3
There is no solution to the 3-Queens Problem
wirelessprvnat-172-16-249-2:Debug sreenivas$ ./N\ Queens\ Problem 8
8-Queens Problem Solution
------------------------
 Q  -  -  -  -  -  -  -
 -  -  -  -  -  -  Q  -
 -  -  -  -  Q  -  -  -
 -  -  -  -  -  -  -  Q
 -  Q  -  -  -  -  -  -
 -  -  -  Q  -  -  -  -
 -  -  -  -  -  Q  -  -
 -  -  Q  -  -  -  -  -
------------------------
wirelessprvnat-172-16-249-2:Debug sreenivas$ ./N\ Queens\ Problem 10
10-Queens Problem Solution
------------------------------
 Q  -  -  -  -  -  -  -  -  -
 -  -  -  -  -  -  -  Q  -  -
 -  Q  -  -  -  -  -  -  -  -
 -  -  -  -  -  -  -  -  Q  -
 -  -  -  -  -  Q  -  -  -  -
 -  -  Q  -  -  -  -  -  -  -
 -  -  -  -  -  -  -  -  -  Q
 -  -  -  Q  -  -  -  -  -  -
 -  -  -  -  -  -  Q  -  -  -
 -  -  -  -  Q  -  -  -  -  -
------------------------------
wirelessprvnat-172-16-249-2:Debug sreenivas$
```

Figure 4: Sample output of the code shown in figure 2.

| $N \times N$ | Total #of Solutions |
|---|---|
| $8 \times 8$ | 92 |
| $9 \times 9$ | 352 |
| $10 \times 10$ | 724 |
| $11 \times 11$ | 2,680 |
| $12 \times 12$ | 14,200 |
| $13 \times 13$ | 73,712 |
| etc. | etc |

Table 1: #Solutions to the $N$-Queens problem as a function of $N$.

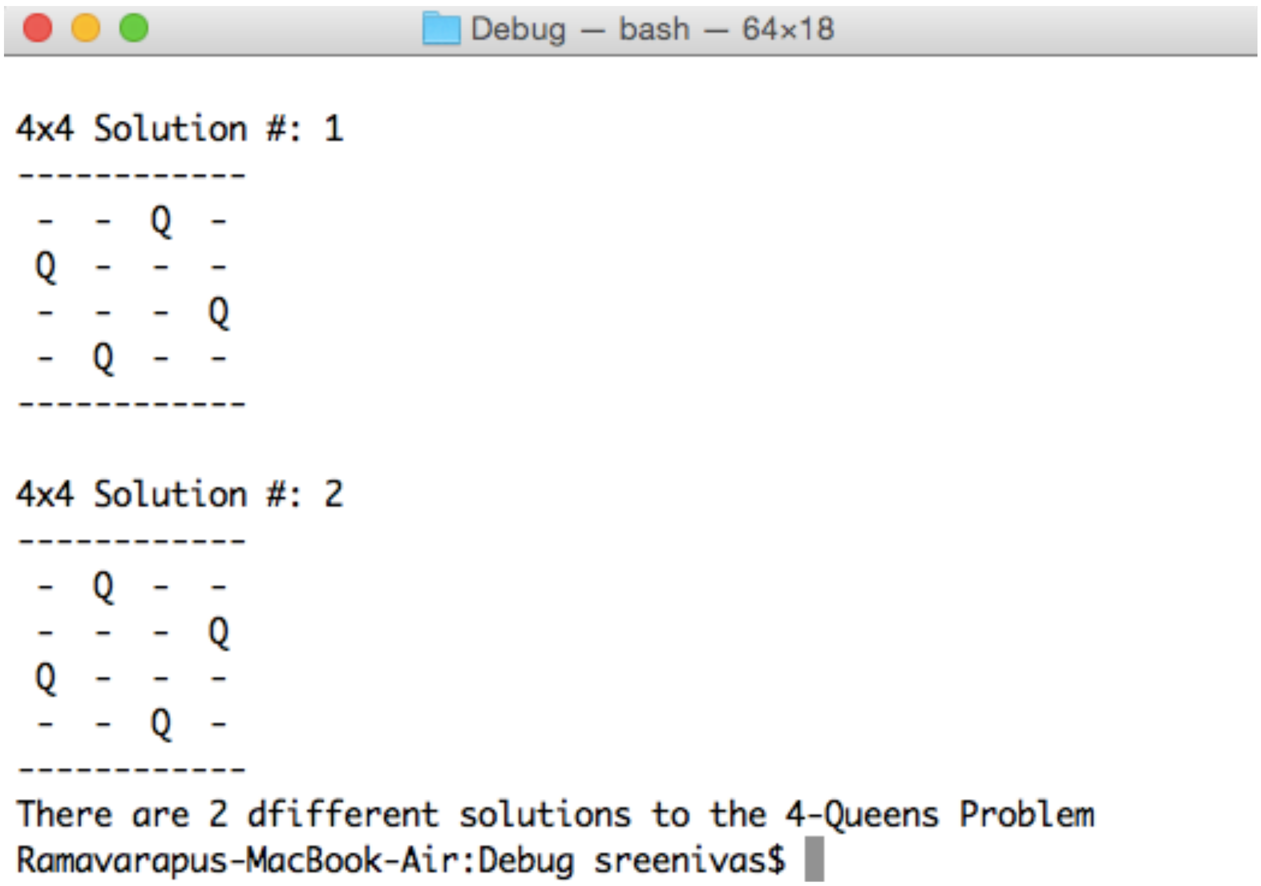



Figure 5: Sample output of the code that computes all solutions to the $N$-Queens Problem. This is for $N = 4$.

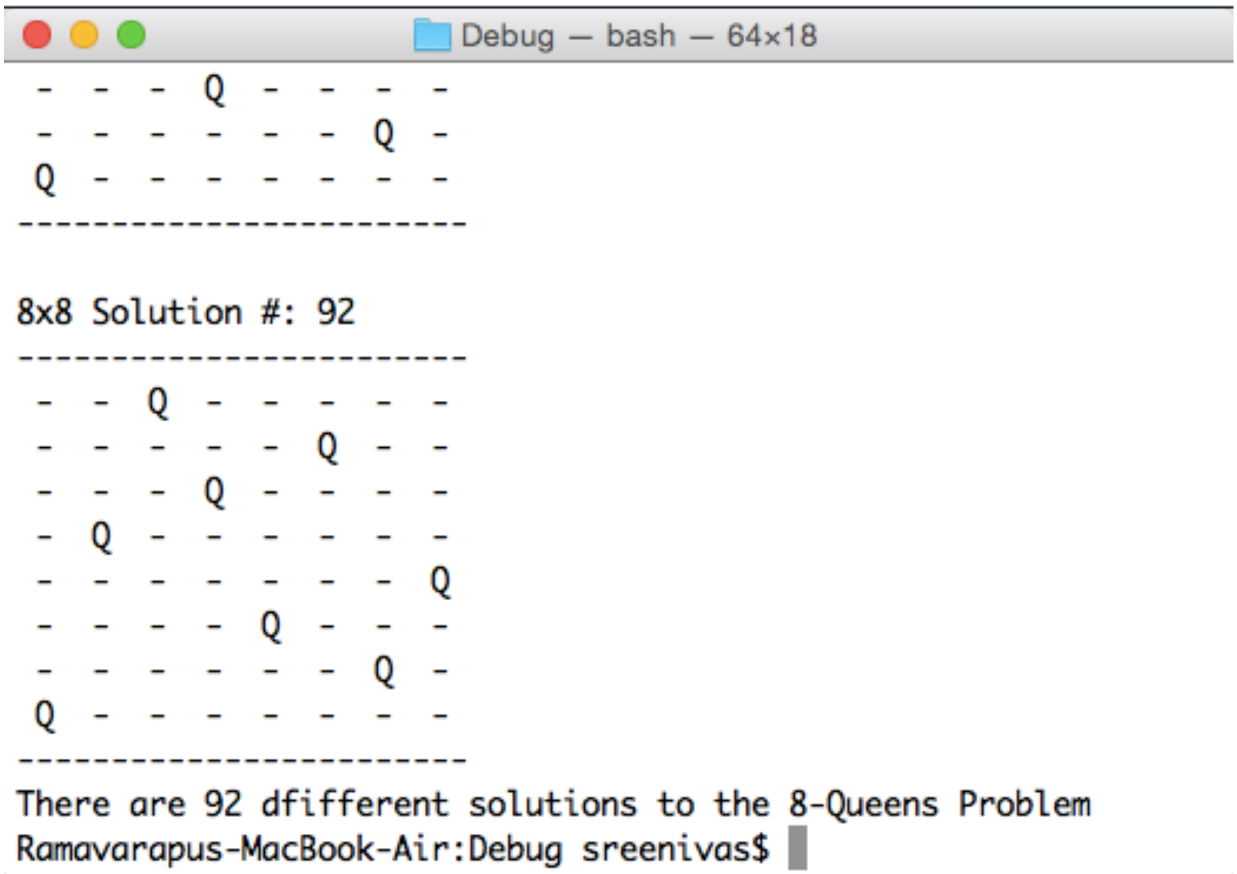
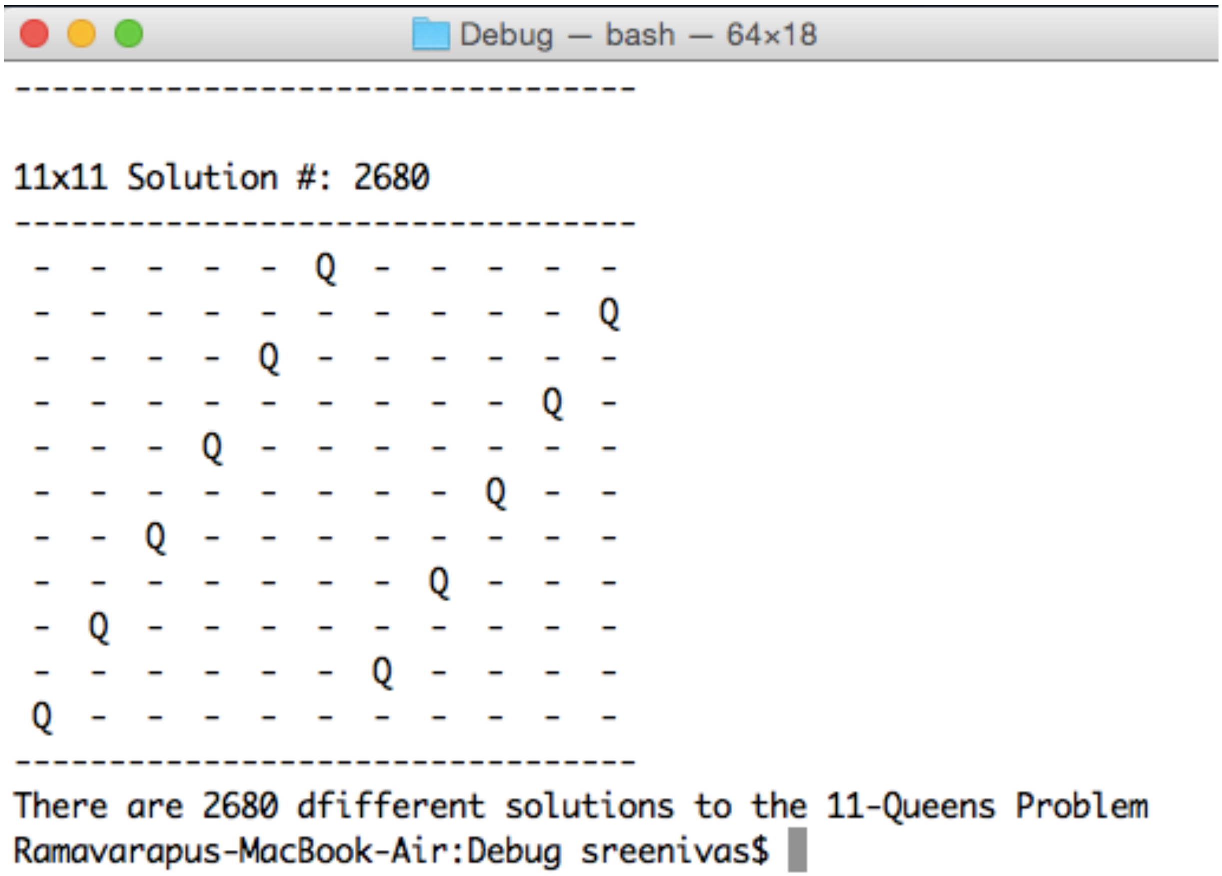



Figure 6: Sample output of the code that computes all solutions to the $N$-Queens Problem. This is for $N = 8$.



```
Debug — bash — 64×18
---------------------------------

11x11 Solution #: 2680
---------------------------------
 -  -  -  -  -  Q  -  -  -  -  -
 -  -  -  -  -  -  -  -  -  -  Q
 -  -  -  -  Q  -  -  -  -  -  -
 -  -  -  -  -  -  -  -  -  Q  -
 -  -  -  Q  -  -  -  -  -  -  -
 -  -  -  -  -  -  -  -  Q  -  -
 -  -  Q  -  -  -  -  -  -  -  -
 -  -  -  -  -  -  -  Q  -  -  -
 -  Q  -  -  -  -  -  -  -  -  -
 -  -  -  -  -  -  Q  -  -  -  -
 Q  -  -  -  -  -  -  -  -  -  -
---------------------------------
There are 2680 dfifferent solutions to the 11-Queens Problem
Ramavarapus-MacBook-Air:Debug sreenivas$
```

Figure 7: Sample output of the code that computes all solutions to the $N$-Queens Problem. This is for $N = 11$.