CS 262, Fall 2014, All sections
Project #1 (100p)

Due date: Wednesday Oct. 1 at 11:59pm

Your first project will be to write a C program to solve the n-Queens problem. The objective is to place n queens on a $n \times n$ chess board: no two queens are allowed to be in the same row, column, or a diagonal.

You can represent a board configuration with a simple n-element array $\{c_0\ c_1\ \cdots\ c_{n-1}\}$; c_i corresponds to the column position of ith queen in row i. For example, {0 2 1 3} would correspond to the 4 queens in positions $(0,0)$ (row 0, column 0), $(1,2)$ (row 1, column 2), $(2,1)$ (row 2, column 1), and $(3,3)$ (row 3, column 3). Note that in our representation we do not use row indexes: the ith queen is placed in the ith row and its column index is c_i.

There are multiple ways to solve this problem. The simplest one would be to generate all possible configurations and check if they solve the problem. Note that there are n^n possible configurations that would need to be checked. You can do better if you realize that you only need to check permutations of numbers $0, \ldots, n-1$ as possible configurations. (Why?) Note that in the case of the 4 queens that would be 256 possible board configurations, but there are only 24 permutations; you can verify that there are just 4 legal 4-queen placements. However, the method for generating all possible permutations is somewhat involved; it is much easier to generate random permutations as was done in Lab. #4.

Your assignment is to write a program to solve n-queens problem for $n = 4, \ldots, 20$ using random board configurations. You should use your *randperm* function from Lab. #4. In addition, you will write two additional functions: *check board* for checking if a permutation solves the problem, and *displayboard* for printing a solution. Finally, for each board size you should solve the problem 10 times so that you can obtain some statistics on performance of your program.

Detailed requirements:

0. You will seed the random number generator using *srandom(seed)*, where *seed* ={last 4 digits of your G#}.

1. You will use your *void randperm(int b[], int n)* size function from Lab. #4 to generate random boards. Note that you only need to initialize *b[]* once for each board size.

2. Write
   int checkboard(int b[], int n)
   checkboard returns 1 if the board represented by *b[]* solves n-queens problem, it returns 0 otherwise.

3. Write *int displayboard(int b[], int n)*
   displayboard prints a solution of n-queens problem in an easy to check form. Here is a possible way your output should look for $n = 6$.

```
[ 4 2 0 5 3 1]
- - - - * -
- - * - - -
* - - - - -
- - - - - *
- - - * - -
- * - - - -
```

4. For each board size $n = 4, \ldots, 20$ you will run your program 10 times to collect some statistics on its performance. For each n you will calculate *min*, *max*, and *mean* number of random boards generated until a solution was found. To calculate *mean* value for any board size you need to add up the total number of boards generated for that size and divide it by 10. To calculate the *min* and *max* values you can utilize the following macros:

```
#define max(a,b) \
       ({ __typeof__ (a) _a = (a); \
           __typeof__ (b) _b = (b); \
         _a > _b ? _a : _b; })
#define min(a,b) \
       ({ __typeof__ (a) _a = (a); \
           __typeof__ (b) _b = (b); \
         _a < _b ? _a : _b; })
```

   The TAs and instructors will explain their purpose and use.

5. You will run your program for board sizes $n = 4, \ldots, 20$ ten (10) times. You will display the first solution only using *displayboard*. You will display the following statistics for each size n: *min, max,* and *mean* number of boards generated before a solution was found, n^n and $n!$. You can use integers to calculate *min* and *max* values but you need to use floats or doubles to calculate the remaining three values. Here is an example output:

size	min	max	mean	size**size	size!
4	1	52	1.6e+01	2.6e+02	2.4e+01
5	1	24	1.1e+01	3.1e+03	1.2e+02
6	9	773	1.5e+02	4.7e+04	7.2e+02
7	1	340	1.1e+02	8.2e+05	5.0e+03
8	45	1508	4.7e+02	1.7e+07	4.0e+04
9	3	2115	9.3e+02	3.9e+08	3.6e+05
10	375	17827	4.3e+03	1.0e+10	3.6e+06
11	109	49104	1.8e+04	2.9e+11	4.0e+07
12	354	158594	4.3e+04	8.9e+12	4.8e+08
13	15554	420771	1.1e+05	3.0e+14	6.2e+09
14	5042	1212410	2.8e+05	1.1e+16	8.7e+10

15	14734	3133599	7.2e+05	4.4e+17	1.3e+12
16	248658	6873571	1.9e+06	1.8e+19	2.1e+13
17	144108	8658763	1.9e+06	8.3e+20	3.6e+14
18	368100	29683930	9.5e+06	3.9e+22	6.4e+15
19	113601	47524247	1.6e+07	2.0e+24	1.2e+17
20	429506	202384581	6.6e+07	1.0e+26	2.4e+18

Instructions for submission:

1. Use script do show your session in which you compile and run your code for various values of n. You should show testing of your functions.

2. Use *tar* or *zip* command to create an archive of your script file, your fully commented source code, and your Makefile.

3. Submit your *tar* or *zip* file on Blackboard.