# CMPS 12B-01, Fall 2018: HW 1
# How to place your queens

- All assignments must be submitted through git. Please look at the Piazza guide on submitting assignments.
- Please follow the naming conventions properly. Please look at the Piazza guide on the checking script. Run the checking script to make sure your files are named correctly. You will get no credit if the checking script fails!
- Follow instructions, and carefully read through the input/output formats.
- Clearly acknowledge sources, and mention if you discussed the problems with other students or groups. In all cases, the course policy on collaboration applies, and you should refrain from getting direct answers from anybody or any source. If in doubt, please ask the instructors or TAs.

## 1 Problem description

**Main objective:** Solve the $n$ queens problems. You have to place $n$ queens on an $n \times n$ chessboard such that no two attack each other. Important: the chessboard should be indexed starting from 1, in standard $(x, y)$ coordinates. Thus, $(4, 3)$ refers to the square in the 4th column and 3rd row.

We have a slight twist in this assignment. We will take as input the position of one queen, and have to generate a solution with a queen in this position.

**Format:** You should provide a Makefile. On running `make`, it should create "NQueens.jar". You should run the jar with *two* command line arguments: the first is an input file, the second is the output file. For example, we would call the command `java -jar NQueens.jar in.txt solution.txt`. The file `in.txt` will have inputs to the main program (as described below), and the file `solution.txt` will have the desired solution.

Each line of the input file corresponds to a different instance. Each line of the input file will have three integers: the chessboard size, the column where the input queen is placed, and the row where the input queen is placed. For example, the file may look like:

7 3 2
4 1 1

The first line means we have a $7 \times 7$ chessboard, with one queen placed at $(3, 2)$. We wish to place 6 more queens without any attacking the other. The

second line means we have a $4 \times 4$ chessboard, with one queen at $(1, 1)$. We wish to place 3 more queens without any attacks. So on and so forth.

**Output:** On running the command, the following should be printed in the output file. For each line of the input file, there is a line in the output file with the placement of queens.

- If there is no solution to the problem, print "No solution" (with a newline at the end)
- If there is a solution, print the position of each queen as `<column> <space> <row> <space>` followed by the position for the next queen. The positions should be in increasing order of column, so first column first, second column second, etc. Please follow this format exactly, so that the checking script works for you.

For example, the output for the input describe above could be

1 1 2 5 3 2 4 6 5 3 6 7 7 4
No solution

Observe how "3 2" is part of the first solution, since we started with a queen there. The solution is not unique, so your code might find some other placement. On the other hand, a $4 \times 4$ chessboard with a queen at $(1, 1)$ has no solution.

**Helper code:** On the HW website, you will find a java file that prints out your solution on chessboard (with queen positions) on to your console. This is extremely helpful in checking if your solution is correct. Compile and execute the java code on terminal directly using the commands below:

- `javac PrintSolutionHelper.java`
- `java PrintSolutionHelper <your output file>`

For each line in your output file that has a list of queens positions, the console will print the chessboard with queens on then. (The code is actually pretty interesting!)

**Example commands and outputs:** On the HW website, you will find a file `Examples.zip`. Download and unzip it. There are four files:

- `test-input.txt, test-output.txt`. If you run your program with input file `test-inputs.txt`, the output file should be exactly the same as `test-outputs.txt`. This will be used by the checking script.
- `more-input.txt, more-output.txt`. The former file has, ahem, more inputs, and the latter file has the outputs. For these instances, the corresponding outputs might not be unique. *So, your solution may be different from what is given, since there can be numerous solutions.*

## 2 Grading

You code should terminate within 3 minutes for all runs (maximum chessboard size will be 14). You get no credit for a run that does not provide a proper output.

Please comment and structure your code properly. You will lose up to 2 points for poor documentation.

1. (12 points) For a full solution as described above.
2. (10 points) Only solves the problem when the input queen is in the first column. (This makes your recursive code a lot simpler to write.) When the input queen is not in the first column, just print "No solution".
3. (8 points) Only solves for chessboards at most $6 \times 6$. You can set up 12 nested loops (6 for rows and 6 for columns) that simply try all possible placements of 6 queens.
4. (6 points) Only solves for chessboards at most $4 \times 4$.