CS262, Project 1:
The *n*-Queens

Due: Sunday, March 13, at 11:59 pm ET

Description:

In this project you will write a C program to solve the *n*-Queens problem.
The goal 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 sequence $Q_n = \{q_0\, q_1 \ldots q_{n-1}\}$; q_i corresponds to the column position of *i*th 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 *i*th queen is placed in the *i*th row and its column index is q_i. To check whether a sequence Q_n solves the *n*-Queens problem you need to make sure that

1. Two queens do not share a column, i.e. $q_i \neq q_j$, for $i \neq j$, and
2. Two queens do not share a diagonal, i.e. $|q_i - q_j| \neq |i - j|$, when $i \neq j$

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,... , $n-1$ as possible configurations. 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.

Your assignment is to write a program to solve the *n*-queens problem for $n = 4, \ldots, 20$ using random board configurations. You should use your *randperm* function to do this. In addition, you will write two additional functions: *checkboard* 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 requiremets:

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

1. You will write *void randperm*(*int b*[], *int n*) function to generate random boards. Note that you only need to initialize *b*[] once for each board size. A simple algorithm for generating a random permutation of elements of a chooses a random number $0 \le i \le n - 1$ and swaps $a[i]$ and $a[n - 1]$; this step is then repeated for the subsequence $a[0 : n - 2]$. The algorithm can be written as

```
Algorithm randperm(int a[], int n)
    for $i = n - 1$ downto 1 do
        $d_i \leftarrow$ random element of $\{0, \ldots, i\}$
        swap $a[d_i]$ and $a[i]$
```

Use *srand*() and *rand*() to generate random numbers. *srand*(1) will initialize your random number generator with seed 1, and $k = rand()$ will return a random long integer k. To produce a random number j such that $0 \leq j < p$ you need a call $j = rand()\%p$. To use *srand()* and *rand*() you will have to include `<stdlib.h>`

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 $n = 4,\dots, 20$ 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.

5. You will run your program for board sizes $n = 4,\dots, 20$ ten (10) times. You will display the first solution only for each value of *n* 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
```

Makefile:

Create a *Makefile* to compile your program and to remove the executable.
Use the same template of the *Makefile* specified on `Lab5 assignment`
The name of the source file will be `p1_<username>_<labsection>.c`

Submitting:

1. On zeus, create a directory named `p1_<username>_<labsection>`. Copy your source file and *Makefile* to this directory.
2. Create a typescript with the following content:
   a. Show that you are on zeus,
   b. Show a listing of your directory
   c. Show your source code
   d. Compile the code using the *Makefile*.
   e. Run your program
3. Be sure your directory ONLY contains the *source file*, *typescript* and *Makefile*
4. Change to the parent directory and create a tarfile of your project directory. Name this tarfile `p1_<username>_<labsection>.tar`
5. Submit this `tarfile` to Blackboard no later than Due Date.

Congratulations! You have completed your Project