CS2100 Computer Programming Lab
Backtracking and Recursion.

Release Date: October 6, 2015
Due Date: October 16, 2015

Problem 1 : Word Sudoku

This is the familiar Sudoku problem using alphabets. We are given a 9×9 grid and a set of English alphabets. There are 9 boxes each containing 9 cells as marked in a usual sudoku. Some of the cells of the grid are prefilled using the alphabets. The goal is to decide whether the 9×9 grid can be completely filled using the given alphabets such that

- Every alphabet appears in each row exactly once.
- Every alphabet appears in each column exactly once.
- Every alphabet appears in a box exactly once.

Use backtracking to implement a sudoku solver.

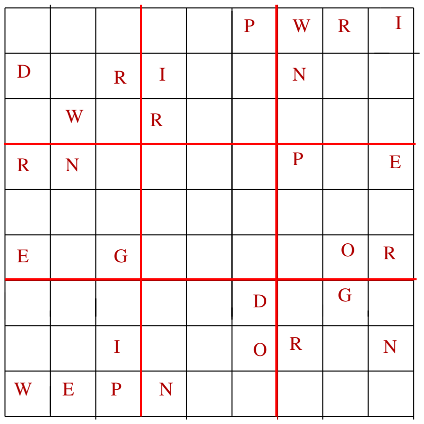
Input Output

Write a program that accepts 2 files as the command line parameters – the input file and the output file. Assume that the input file has the following format. The first line contains the 9 distinct alphabets separated by a comma. The next nine lines specify the sudoku prefilled cells. Each line is a comma separated line, where unfilled cells are represented using '-'.

For example, the input file given below represents the sudoku shown beside.

sudoku-input.txt

```
P,O,W,D,E,R,I,N,G
-,-,-,-,-,P,W,R,I
D,-,R,I,-,-,N,-,-
-,W,-,R,-,-,-,-,-
R,N,-,-,-,-,P,-,E
-,-,-,-,-,-,-,-,-
E,-,G,-,-,-,-,O,R
-,-,-,-,-,D,-,G,-
-,-,I,-,-,O,R,-,N
W,E,P,N,-,-,-,-,-
```



sudoku-output.txt

```
N,G,E,O,D,P,W,R,I
D,P,R,I,G,W,N,E,O
I,W,O,R,N,E,G,D,P
R,N,D,G,O,I,P,W,W
P,O,W,D,E,R,I,N,G
E,I,G,W,P,N,D,O,R
O,R,N,P,I,D,E,G,W
G,D,I,E,W,O,R,P,N
W,E,P,N,R,G,O,I,D
```

Your output should be a file containing a solution to the sudoku, if it exists. The solution should contain 9 lines each of which contains 9 alphabets separated by a comma. One valid output for the given sudoku is as shown above. If the given sudoku does not have a solution, output in the file "No solution".

Problem 2 – A game of tokens between two players [1]

Consider a simple two player game played on an $n \times n$ grid. The two players Player-1 and Player-2 start with n tokens. Player-1 places his tokens vertically just outside on the left edge of the board. Player-2 places his tokens horizontally just outside at the top edge of the board. Player-1 always moves his tokens horizontally and Player-2 moves his tokens vertically. The goal of each player is to move all his tokens out of the board to the other side. If Player-1 has one of its tokens in the (i, j)-th cell, then the valid moves are:

- If the cell $(i, j + 1)$ is empty, move token to $(i, j + 1)$.
- If the cell $(i, j + 1)$ is occupied with a token of Player-2 and the cell $(i, j + 2)$ is empty, move the token to $(i, j + 2)$.
- Pass if no legal move is possible.

Symmetrically define the moves for Player-2. The first player to get all his tokens out of the board on the other side, is the winner of the game. See Jeff Erickson's notes uploaded on moodle for a detailed example.

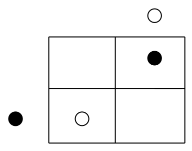
Given a state of the game in terms of the position of the tokens and the current player who is supposed to make the move, we say that the state is a winning position for the current player, if there exists a move that leads to a winning state for the current player irrespective of the moves of his opponent. Our goal in this problem is to determine if the given state for a current player is a winning position.

Input Output

Write a program that accepts 2 files as the command line parameters – the input file and the output file. The input is specified as follows: The first line of the file specifies the dimension n of the problem. The next 2 lines give the positions of the tokens for Player-1 and Player-2 respectively. The position of Player-1 is given in terms of n column values in which the n tokens of Player-1 are present. The column values range from 0 to $n+1$, positions 0 and $n+1$ both being outside the $n \times n$ grid. Similarly, the position of tokens for Player-2 is specified using the n row values in which the tokens of Player-2 are present. Assume that each line has comma separated values. Finally, the last line of the file is either P1 or P2 and denotes the player who is supposed to play next. For example, input file given below shows the position of the game as show in the Figure. We use the dark marked circles to denote tokens of Player-1.

game-input.txt

```
2
2,0
2,0
P2
```

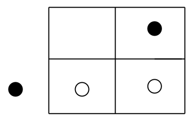


Your program should decide whether the given position is a winning position for the player specified. If yes, it must output the state of the game obtained by making any one of the winning moves for the given player. If the position is not a winning position, the output should be "No winning move".

For example, in the above given position, there are exactly two moves for Player-2. Only one of them is a winning move and since such a move exists, the given input position is a winning position for Player-2. The state of the game after making the winning move is as shown below.

game-output.txt

```
2
2,0
2,2
P1
```



Verify that the other move (from the configuration in game-input) is not a winning move for Player-2 since Player-1 can force a win for himself from the resulting position.

[1]Courtesy Jeff Erickson's lecture notes

Submission Guidelines

Follow the usual submission guidelines and submit two different files containing the codes for Problem1 and Problem2. Remove all temporary files and object files and make a tar.gz and upload the tar ball to moodle.