CS1132 Spring 2016 Assignment 2 Due Apr 20th

Adhere to the Code of Academic Integrity. You may discuss background issues and general strategies with others and seek help from course staff, but the implementations that you submit must be your own. In particular, you may discuss general ideas with others but you may not work out the detailed solutions with others. It is never OK for you to see or hear another student's code and it is never OK to copy code from published/Internet sources. If you feel that you cannot complete the assignment on you own, seek help from the course staff.

When submitting your assignment, follow the instructions summarized in Section 4 of this document.

Do not use the `break` or `continue` statement in any homework or test in CS1132.

1 Contact information retrieval system

In this problem, you will implement a contact information retrieval system, which reads contact information from an input text file, allows the user to query contact information by name, and writes the query history to an output text file.

Implement a function as specified below. We'll explain the components of this system and subfunctions you need to write in the following sections.

```
function retrieveContactInfo(inputFile, outputFile)
% A contact information retrieval system. This function reads the contact
% information from a text file and save the query log to another text file.
% The user can enter the name to query the information of a contact, or
% enter ':q' to quit the system. If the input name matched some contacts in
% the data, the information of all matched contacts is printed. Otherwise
% the system finds contacts who have similar names and print their information.
% inputFile: name of input text file containing the contact infomation.
% outputFile: name of the output query log file.
```

1.1 Input file

The input text file contains the contact information. In this file, each line represents a contact. Each line of text can be split into several substrings, and two neighboring substrings are separated by a separator `'; '` (a semicolon followed by a space). There is no separator at the beginning or end of the line. In each line, the first substring is the contact name. The contact name can be followed by a list of substrings, each represents additional contact information, such as numbers and address. Implement the following subfunction that reads the input file and returns a cell array of contact information, as specified.

```
function data = readContactFile(fn)
% Read contact information from file.
% fn: the input file name.
% data: information of all contacts, a 1-d cell array of cell arrays, where
%   data{i}{1} is the name of the ith contact,
%   data{i}{2}, if it exists, is additional info of the contact,
%   data{i}{3}, if it exists, is additional info of the contact,
%   ..., etc.
```

In order to split each line of text, implement the following subfunction, and use it in `readContactFile`.

```
function data = splitString(str)
% Split a line of string into substrings. Two neighboring substrings are
% separated by '; ' (a semicolon followed by a space).
% str: the string to be split.
% data: a 1-d cell array in which each cell is a substring. The separators
% are not included.
```

To split the string, you should first find the starting indices of all separators, and then split the string accordingly. You can **NOT** use the built-in function `strfind`. Instead, you should implement a subfunction as follows, and use it in `splitString`.

```
function indices = myStrFind(str, pattern)
% Find a substring within another string.
% str: an input string.
% pattern: the substring to be found in str.
% indices: the returned 1-d array, which is the starting indices of any
% occurences of the string pattern in the string str.
```

1.2 User interface

The user interface should allow the user to query contact information by entering a name. Use the code below to get an input string from the command window.

```
input('Please enter the contact name (or :q to quit): ', 's')
```

Here the argument `'s'` let `input` return a string instead of a number; the user needs only type the string of interest, without the quotation marks.

Once the user enters a name, the system should find all contacts that have the same name as the input and list their information neatly. The matching should **NOT** be case-sensitive, which means that an input of `'dAvid'` should match a contact `'David'` in the database (`strcmpi` is good at comparing two strings ignoring cases). If the input does not match any name in the database, the system should display a message saying so and then try to find similar contacts (see Section 1.3).

The user can keep querying until he/she enters the quit command `':q'`.

1.3 Similar strings

When the user enters a string that does not match any name in the database, the system should find every contact who has a similar name to the queried name. We measure similarity between two strings using Levenshtein distance (read the corresponding Wikipedia page if you are interested in this metric), which is the minimum number of single-character edits needed to change one string into the other. The system should print the information of all contacts whose name has a strictly less than 3 Levenshtein distance from the queried string. If no similar name is found, print a message. Again this should **NOT** be case-sensitive (the built-in functions `lower` or `upper` will be useful). We provide you the function `strDist` to compute the Levenshtein distance between two input strings, which is case sensitive.

1.4 Output file

The system should write the query history to the output file. The output is a text file, in which each line represents one query by the user. Each line of the file should have the following format, which starts with the time when the queried is made and ends with the string entered by the user. The `':q'` command should not be recorded in the log. Use built-in function `clock` to get the current time.

```
02/25/2016 14:33:59 davi
```

1.5 Built-in functions

Below is a list of useful built-in functions for handling characters, strings and files. ***Use only these built-in functions*** for handling characters, strings and files. You may not need all of them. You can of course still use general built-in functions not related specifically to strings and files, such as `length`, `zeros`, `cell`, etc.

You can use:

- `fopen`, `fclose`, `feof`, `fgetl`, `fprintf`.
- `strcmp`, `strcmpi`, `lower`, `upper`.

You must **NOT** use MATLAB's built-in functions `find`, `strfind`, `findstr`.

Also, you should write your own subfunctions, whenever necessary, to avoid redundant code and make your code more readable.

2 Interactive Reversi game

In Assignment 1b, you've implemented some logics and an AI for the Reversi game. In this assignment, you're going to make an actually playable Reversi game using MATLAB's interactive graphics. Implement the follow function as specified. Take a look at the video to see how the game should look like.

```
function reversiGame()
% An interactive Reversi game.
% The user can left click the mouse to place a black disk or right click
% the mouse to temporarily see the board state after placing a black disk
% at that position. The white disks are placed by an AI player with a
% greedy strategy. Whenever there is no legal position for a certain color,
% the corresponding player's turn is skipped. The game ends when no disk
% can be legally placed on the board. The player who has more disks of
% his/her color on the board wins the game.
```

The `reversiGame` should support the following interactions.

- The game starts with 2 white disks and 2 black disks and draws the board in a figure, the same as is in Assignment 1b. The user plays black and the AI plays white. The black moves first.
- The user can click on the board to play the game. To place a new black disk, LEFT click on a square. If the position is legal, the black disk is placed and the board is updated. If not, a red cross is drawn on that square and a message is displayed in the title to indicate that the click was illegal.
- The user can RIGHT click to temporarily see how the board will be updated after placing a black disk at a certain position. If it's a legal position, the window shows the board after placing the new disk for a short time (e.g. 0.5s) and then returns to the original state. A message indicating the number of reversed disks is shown in the title. If it's an illegal position, a red cross is drawn on that square and a messages is prompted in the title. Use the following line to pause the program.

  ```
  pause(0.5);  %pauses program execution for 0.5 seconds
  ```

- After the user places a new black disk on the board, the AI then places a new white disk on the board, if there exists a legal position for white, such that the number reversed disks is maximized. The reversed disks twinkles after a white disk is placed (see the video).
- In either the AI or the user's turn, if there is no legal position for the corresponding color, this turn should be skipped and the opponent should move next. This means that after the AI places a white disk, you need to check whether there is a legal position for black; if not, the AI should continue to place another white disk. Whenever the user's turn is skipped, display a message in the title.

- Whenever a new disk is placed on the board, the title should show the current number of black and white disks respectively.
- When there is no legal position for both black and white, the game ends and the title shows which color has won the game.

Here are some requirements for implementing this function.

- Use `ginput` to get the position of the cursor and the clicked button: `[x,y,b]=ginput(1)` returns in `x` and `y` the x- and y-coordinates of the user's click and in `b` the mouse button used, numbered 1, 2, and 3 from the left (therefore 1 indicates a left click and 3 indicates a right click).
- Use `drawBoard` from assignment 1b to draw the grid and disks.
- Make effective use of your `placeNewDisk` and `pickNewPosition` from assignment 1b, to update the board state, place the disks and determine the existence of legal positions.
- Use vectorized logical operations and `sum`, instead of loops, to determine the number of black/white disks on the board. Be sure that you learn how to do this; it's an important operation in MATLAB and we will ask you about it on the test.
- Write subfunctions to avoid redundant code and make your code more readable.

3 Self-check list

The following is a list of the minimum *necessary* criteria that your assignment must meet in order to be considered *satisfactory*. Failure to satisfy any of these conditions will result in an immediate request to resubmit your assignment. Save yourself and the graders time and effort by going over it before submitting your assignment for the first time.
Note that, although all of these are necessary, meeting all of them might still not be *sufficient* to consider your submission satisfactory. We cannot list everything that could be possibly wrong with any particular assignment!

- Δ Comment your code! If any of your functions is not properly commented, regarding function purpose and input/output arguments, you will be asked to resubmit.
- Δ Suppress all unnecessary output by placing semicolons (;) appropriately. At the same time, make sure that all output that your program intentionally produces is formatted in a user-friendly way.
- Δ Make sure your functions names are *exactly* the ones we have specified, *including* case.
- Δ Check that the number and order of input and output arguments for each of the functions matches exactly the specifications we have given.
- Δ Test Each one of your functions independently, whenever possible, or write short scripts to test them.
- Δ Check that your scripts do not crash (i.e., end unexpectedly with an error message) or run into infinite loops. Check this by running each script several times in a row. Before each test run, you should type the commands `clear all; close all;` to delete all variables in the workspace and close all figure windows.

4 Submission instructions

1. Upload files `retrieveContactInfo.m` and `reversiGame.m` to CMS in the submission area corresponding to Assignment 2 in CMS.
2. Please do not make another submission until you have received and read the grader's comments.

3. Wait for the grader's comments and be patient.
4. Read the grader's comments carefully and think for a while.
5. If you need to resubmit, fix all the problems and go back to Step 1! Otherwise you are done with this assignment. Well done!