CS1132 Fall 2014 Assignment 2 due 4/16 11:59pm

> *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 Bubble Sort with Index

In Assignment 1a, you implemented bubble sort with the output being the sorted vector. However, in many applications, it is useful to know the index of each sorted element in the original input vector. For example, if the input vector is [4,3,10,5,1], then the output sorted vector is [1,3,4,5,10] and the index vector is [5,2,1,4,3]. One assumption is that the input argument is either a row or a column vector.
Implement this new bubble sort in the following function:

```
function [sortedArray,indexArray] = newBubbleSort(V)
% This function implements bubble sort.
% Input: V is a row or column vector .
% Output: sortedArray is the sorted array after the bubble sort
%         indexArray stores the index location for each element in V
```

2 Consecutive Vowels in Dictionary

A subset of words and phrases from a dictionary have been scrambled and are stored in no apparent order in the text file *randPermDic.txt*. Fortunately, each word/phrase has a rank (entry number) from the dictionary that has been retained in the file. Below are the first few lines from the file:

```
45057,laudanum
24653,common room
23755,clerestory
71146,unrestrained
```

Write a program to read *randPermDic.txt*, identify the words/phrases that contain two or three consecutive vowels, sort those words/phrases by their dictionary rank, and write them to a plain text file in sorted order. Below are the requirements for your solution:

- Implement this function to solve the problem:

```
function n = sortDicWithMultipleVowels(dicName, sortedDicName)
% dicName names the file that contains words/phrases in scrambled order.
% Identify the words/phrases that contain two or three consecutive vowels,
% sort them in ascending order by the word/phrase rank, and write them in
% sorted order to a plain text file.
```

```
% n is the number of lines written to the output file.
% sortedDicName names the output file to be written.
% Eg., n=sortDicWithMultipleVowels('randPermDic.txt','sortedDicWithVowels.txt')
```

- Use a cell array to store the string data from the file. The word/phrase on the first line of the file should be stored in the first cell of the cell array, the word/phrase of the second line of the data file should be stored in the second cell of the cell array, and so forth. The cell array should be just big enough to store the data; there should not be any empty cells. Store the ranks in a corresponding fashion but use a simple array (so that you can later sort the ranks easily).
- Use only the letters 'a', 'e', 'i', 'o', and 'u' as vowels (include both upper and lower case). Filter the original cell array to store the words/phrases with two or three consecutive vowels in a new cell array. Correspondingly you need to keep track of the ranks.
- Use subfunctions! For example, a subfunction that checks one string for two or three consecutive vowels is super useful. Another useful subfunction is one that checks whether a single character is a vowel.
- Use your function `newBubbleSort` from Problem 1 to sort the ranks. Sort in ascending order.
- Write the words/phrases that have two or three consecutive vowels, including their rank, in sorted order in a plain text file. Part of the resulting file should look like this:

```
932,Anzio
933,Aorangi
936,Apeldoorn
940,Apia
946,Apollinaire
949,Apollonian
950,Apollonius
952,Apostles' Creed
954,Apostolic See
955,Appalachia
```

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, rem`,$\cdots$ etc.

You can use:

- `fopen, feof, fgetl, fclose`
- `strcmp, upper, isspace, isletter, str2num`

Function `str2num` converts a string representing a number to the numeric value. For example: `str2num('0035')` returns the numeric scalar 35. You must NOT use MATLAB built-in function `find, strfind, findstr, regexp`.

3 Graphics Fun!

In this part you will implement `Minesweeper` using Matlab's graphics functions. For our game, we will consider an $n \times n$ board with m mines randomly distributed among the squares on the board. Initially, all the squares on the board are covered. The goal of the game is to mark all the mines and uncover all the squares that do not contain mines. During the game, the player uses a left click to uncover a square (that hopefully doesn't contain a mine) and a right click to indicate a mine.

With this exercise, you have the freedom to structure the code as you see fit. You should define helper functions as needed to make your solution modular—do not submit one long giant function. The helper functions, i.e., subfunctions, must be saved in the same file containing the main function for the game: `minesweeper.m`. The function takes as input arguments n and m, the dimension of the board and the number of mines, respectively. For scaling purposes, you may assume that n is an integer between 5 and 20. Also, you should make sure that $0 < m < n^2$. Here are the requirements:

- The interaction happens only in one figure window.
- Display the $n \times n$ board in the figure window and the name of the game as the title of the figure
- Pick the location of the m mines randomly with equal probability, i.e., each square on the board is equally likely to have a mine.
- The player can left or right click on a square on the game board. Use the Matlab built-in function `ginput` with an extra output parameter, `button`, to get the button identifier in addition to the coordinates of the user's click. `[x,y,button] = ginput(1)` returns the x-coordinate, the y-coordinate, and the button or key designation for one click. *button* is an integer indicating which button was used: 1 for left, 2 for middle, 3 for right. In our game, you can assume that the player never middle clicks.
- Player clicks on a square that is covered (has not been clicked before): A right click marks the square as a mine and the program should put an X on the square, regardless of whether it really is a mine. A left click ends the game with the player losing if the square contains a mine; otherwise the color of the square is changed and the number of neighboring mines (a number between 0 and 8) is displayed on the square.
- Player clicks on a square that is marked by a number: Prompt the user (using the title) to click another square.
- Player clicks on a square that is marked by an X: A right click does not change the state of the square. Prompt the user to click another square. A left click ends the game with the player losing if the square contains a mine; otherwise the color of the square is changed and the number of neighboring mines is displayed on the square.
- If the player loses (left clicks on a mine), the game must not allow another click. Display a message as the figure title saying that the player has lost and then end the game.
- The player wins the game when all the mines are marked with X and all the empty space are marked with numbers. Display a congratulatory message as the figure title and then end the game.
- If all the squares have been marked but the player has not won (there are more X marks than mines), the game continues—keeps accepting another click.

4 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. Although all these criteria 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! Make sure your functions are properly commented, regarding function purpose and input/output parameters.
- 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 parameters for each of the functions match 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 your script several times in a row. Before each test run, type the commands `clear all; close all;` to delete all variables in the workspace and close all figure windows.

5 Submission instructions

1. Upload files `newBubbleSort.m`, `sortDicWithMultipleVowels.m`, and `minesweeper.m` to CMS in the submission area corresponding to Assignment 1a in CMS before the deadline. Late submission is accepted up to 24 hours after the deadline with a 10% penalty.
2. *After grading:* If you resubmit the assignment, upload your corrected files and *be sure to select the* ***Regrade Request*** *option.*