Notice

This homework will require you to create functions that output text files. Text files cannot be checked against the solution output using `isequal()`, but there is another function you can use to compare text files called `visdiff()`. Suppose your output file is called `'outputFile.txt'` and the solution function produces the file `'outputFile_soln.txt'`. You will have to run both your function *and the solution function* with the same inputs to generate these files. Then use the `visdiff()` function to compare your output with the solution function's output. From the Command Window, type and run the following commands:

```
>> functionName(in);
>> functionName_soln(in);
>> visdiff('outputFile.txt','outputFile_soln.txt');
```

At this point, a new window will pop up. This is the MATLAB File Comparison Tool. It will not only tell you if the selected files match, but it will also tell you exactly what and where all of the differences are. Use this tool to your advantage. Please note that sometimes the comparison will say, "No differences to display. The files are not identical, but the only differences are in end-of-line characters." Do not be alarmed if you see this; you will still receive full credit. Your code should **not** yield extra blank lines in output files or you will lose points.

Please keep in mind that your files must be named exactly as specified in the problem descriptions. The solutions will output files with '`_soln`' appended before the extension. Your output filename should be identical to the solution output filename, excluding '`_soln`'. Misspelled filenames will result in a score of 0. You will still need to use `isequal()` to compare non-text-file function outputs.

MATLAB has lots of additional functions for reading/manipulating low-level text files, but because we want you to learn the fundamentals of low-level file I/O, **we have banned `fileread()` and `textscan()` for all problems on this homework.** The use of either of those functions on any problem will result in a 0 for that problem.

Lastly, please remember to **close all files that you open**. The use of `fclose('all')`, `fclose all`, `fopen('all')`, `fopen all`, or `close all` is not permitted, so make sure to close each file individually. Performing `fclose(fh)` in a loop (or something similar) is considered equivalent to performing `fclose('all')` and is also not permitted. If files remain open after your code finishes running a test case during grading, **you will only receive half credit**, even if your outputs are correct. Regrades will not be accepted for otherwise correct code that fails to close all files.

Happy Coding!
~Homework Team

Function Name: `kerchoo`

Inputs:

1. (*char*) Filename of the sneezed scene script

File Outputs:

1. A .txt file with the fixed scene script

Topics: (*file reading*), (*file iteration*), (*file writing*)

Background:

After a week-long detour in the beautiful town of Radiator Springs, Lightning McQueen FINALLY arrives in California, where he must face Chick Hicks for the first time since their tongue tie with The King. Lightning McQueen rolls out of Mack's trailer and is suddenly, aggressively, and blindingly 'KA-CHICKA'ed right in his eye! Lightning can't help but violently sneeze from the shock of direct light on his optic nerves. Instead of 'KACHOW'ing like usual, he 'KERCHOO's and messes up the script he's been studying in preparation for his acting debut. Now, he needs your help to get the lines back in order before Harv schedules his big audition!

Function Description:

Given the filename of the scene that Lightning McQueen 'KERCHOO'ed (sneezed) all over, read in the file and determine where each line is supposed to go. Every line in the file (even lines with no words) will start with a number corresponding to its pre-sneeze position. You must sort each scene's script based on these line numbers in ascending order, then write the lines to a new file. The new filename should follow the format: `'<filename>_fixed.txt'`. Lightning needs to nail this audition, so make sure you do not alter the lines in any way (other than reordering) when writing them to the new file!

Example:

`cupScene.txt`

```
1 | 7 He did what in his cup?
2 | 2 but I wish I could have.
3 | 5
4 | 1 Lightning McQueen: No,
5 | 4 He won three Piston Cups!
6 | 6 Mater: [spits out on his drink]
7 | 3 They say he was amazing!
8 |
```

```
kerchoo('cupScene.txt')
```

cupScene_fixed.txt

```
1 | 1 Lightning McQueen: No,
2 | 2 but I wish I could have.
3 | 3 They say he was amazing!
4 | 4 He won three Piston Cups!
5 | 5
6 | 6 Mater: [spits out on his drink]
7 | 7 He did what in his cup?
8 |
```

Notes:

- The line numbers will always start at 1 and count in the positive direction.
- There is guaranteed to be exactly one space between the line number and the rest of the line.
- All lines in the input .txt files will end with a new line character. This means there will be a new line character at the end of both the input and output files.

Hints:

- Try storing the lines in a sortable data structure!

Function Name: `carsCarsCars`

Inputs:

1. *(char)* The name of a .txt file containing sentences about the Cars franchise

File Outputs:

1. The .txt file with the corresponding Cars adjustments

Banned Functions:

`strrep, replace`

Topics: (*Low Level I/O*), (*strtok*), (*Parsing*)

Background:

It's National Cars Movie Day!!(not really but everyday should be) So you and your friends go out to find all the news about the cars movie trilogy. However, the more you dive into it you realize that nothing can top the original Cars movie. So you decide to make sure the world realizes the best and only viable one being just Cars and go change any mention of the other two movies to be multiple versions of just Cars.

Function Description:

Given the filename of a .txt file, parse through every line and replace every instance of the words `'Cars2'` or `'Cars3'` with the phrase `'Cars Cars'` or `'Cars Cars Cars'` respectively. Output each corrected line to a new text file named `'<file>_corrected.txt'`.

Example:

`Cars_Movie.txt`

```
1| Cars is cool and the best
2| Cars2 is alright but still not as good
3| Ok what on earth is Cars3
4|
```

```
cars = 'Cars_Movie.txt';
carsCarsCars(cars);
```

`Cars_Movie_corrected.txt`

```
1| Cars is cool and the best
2| Cars Cars is alright but still not as good
3| Ok what on earth is Cars Cars Cars
4|
```

Notes:

- There may be occurrences of Cars films that are greater than 3 and those don't exist in the Pixar world so we don't care about changing them.
- All instances of Cars referring to the movie will have a capitalized 'C' so other instances are not significant.
- There will be no spaces between the word 'Cars' and the number if there is one (you will only every have `'Cars2'` and **not** `'Cars 2'`)
- You will have a single newline at the end of each file.
- You must not have a space at the end of your
- There will be no punctuation.

Function Name: `iAmSpeedRead`

Inputs:

1. (*char*) The name of a .txt file containing the script you are reading

Outputs:

1. (*char*) A string describing the reading

Topics: (*Low Level I/O*), *(strtok)*, (*iteration*)

Background:

You are an actor applying for a role in Disney's new live-action remake of Cars. To practice for your audition you read through the script at varying speeds and you want to use MATLAB to check if you're doing so correctly.

Function Description:

Given the filename of a text file, open that file and determine how long it would take you in seconds to read the file. You start reading at a speed of 2 words/second, but whenever you read the word `'Ka-chow'` your speed doubles after you finish reading the word, and it halves after reading `'Ka-chicka'`. Output a string describing how the reading went, according to the following format:

```
'It took <Number_of_seconds> seconds to read the script and I finished reading at <reading_speed_at_end_of_file> words/second.'
```

Example:

`script.txt`

```
1| Cars
2| Cars Cars
3| Ka-chow
4| Word word word word
```

```
str = iAmSpeedRead('script.txt')
str → 'It took 3.00 seconds to read the script and I finished reading at 4.00 words/second.'
```

Notes:

- Both `'Ka-chow'` and `'Ka-chicka'` should be case sensitive, but ignore any punctuation attached to a word besides the hyphen.
- Round number of seconds and final reading speed to two decimal places.
- The test cases are long, so if you have having trouble debugging, try creating your own shorter text files by truncating the test cases to have only 15 or so lines to start and then adding more lines as you need to test longer and longer text files
- For this problem, a number of type char is counted as a word (e.g. 5 and 46 are both considered a word for the purposes of counting how long it takes you to read)

Function Name: `pistonCup`

Inputs:

1. (*char*) Filename of a text file containing other filenames

Outputs:

1. (*struct*) 1xN structure vector with all the car's stats

Topics: (*iterative file reading*), (*string parsing*), (*strtok*), *(structures)*

Background:

Wowee, you've been tasked to report on this year's Piston Cup! Because it's been a while since you've seen the last Cars movie, you need to brush up on the stats of each racer. You ask your boss for a list of the racers, to which she just hands you a list of all the types of cars... super helpful, thanks boss. Unfettered by this, you whip out MATLAB and get to work!

Function Description:

Given the filename of a text file that contains more filenames, where each line of the original text file is the name of another filename. These filenames will always follow the following format:

`<anything>_<type>.txt`

Locate and open **only** the file with **one underscore**, followed by the `<type>` `'cars'`, **case sensitive**. Only one filename in the original file meets this requirement. This file contains N filenames. Open each of the N files. Each inner file will have a single line with the fieldnames and values in the following format:

`<fieldname1>:<value1>,<fieldname2>:<value2>,...<fieldnameM>:<valueM>`

The same fieldnames are guaranteed to appear in each of these inner files, but their order in the line may vary. Construct a 1xN structure vector from the N files and the corresponding fieldnames and values, where each structure in the structure vector contains the information from a singular inner file.

Example:

`vehicleTypes.txt`

```
1| mean_ricers.txt
2| pit_crew.txt
3| not___cars.txt
4| alsonot_CARS.txt
5| fast_cars.txt
6| grazing_tractors.txt
```

fast_cars.txt

```
1| lightning.txt
2| chickHicks.txt
```

lightning.txt

```
1| Name:Lightning McQueen,Ranking:1,Speed:9001,Arrogant:At first
yea
```

chickHicks.txt

```
1| Speed:95,Arrogant:Definitely,Ranking:3,Name:Chick Hicks
```

```
carStats = pistonCup('vehicleTypes.txt')
```

carStats →

Name:	'Lightning McQueen'	'Chick Hicks'
Ranking:	'1'	'3'
Speed:	'9001'	'95'
Arrogant:	'At first yea'	'Definitely'

Notes:

- Keep all values as type char, even the numerical values!

Hints:

- Recall that a structure's field names have no particular order.
- `strtok()`'s second input allows you to specify a delimiter.

Function Name: `tractorTipping`

Inputs:

1. (*char*) The filename of the map of the tractor field

Outputs:

1. (*char*) String describing outcome of Mater's tractor tripping escapade

File Outputs:

1. A .txt file containing the modified map

Topics: (*line-by-line reading*), (*cell arrays*), (*iteration - pathing*), (*masking*)

Background:

You are Floyd Van Diesel, the hottest new racing analyst in town. You are visiting the quaint town of Radiator Springs, where you meet Lightning McQueen and his best friend, Tow Mater. Mater decides to introduce you to one of his favorite pastimes: tractor tipping (https://youtu.be/J4-7Kvhbthg). You immediately become obsessed with the activity and you think you can help bring the activity to national prominence. Of course, you have to make it a little safer by finding a way to avoid Frank while maximizing the number of tractors tipped. Luckily, you have MATLAB and Mater to help you figure it out!

Function Description:

Given the filename of a text file that contains a map of the area around Radiator Springs. There are a few special tiles to keep in mind:

- `'M'` indicates your starting position
- `'E'` indicates where your final position should be (the exit)
- `'>', '<', '^',` and `'v'` indicate a direction change
  - `'>'` directs you right
  - `'<'` directs you left
  - `'^'` directs you up
  - `'v'` directs you down (lower-case v)
- `'F'` indicates the location of Frank
- `'T'` indicates a tractor at that location
- `'.'` is an empty tile with no special attributes (continue going in the same direction if you come across this tile)

Begin by finding your initial position. Your "vision field" (VF) corresponds to the tiles immediately above, below, and to either side of your current position. Likewise, Frank's VF corresponds to the adjacent tiles around him. Look for your initial direction by looking at your initial VF (highlighted red in the example) for direction-changing tiles. There will be **only one** direction-changing tile in your initial VF. Proceed by going in that direction.

As you iterate through the map:

- At every location you've been at, replace the character at that location with a `'#'`. This should represent the path you will take on your journey.
- Change directions **when you land on** a direction-changing tile. These can only be used one time to change your direction and should be replaced by a `'#'` after being traversed.
- For each new `'T'` within your VF, add 1 to the total amount of tractors you have tipped. Note that your current position tile is not included in the VF (see notes).
- If at any point you enter Frank's VF, you get caught. At this point, replace your location with a `'C'` instead of a `'#'`.
- If you reach the exit, do not tip any tractors in your VF when at that location.

Depending on how you finished, you will output 1 of the 3 following possible strings:

- If you are caught, output the string:

```
'You tipped <# tractors tipped> tractors before being caught by
        Frank. You'd better run or he's gonna get you!'
```

- If you tip all the tractors and make it all the way to the end, output the following string:

```
'Yay, you tipped all <total # of tractors> tractors!! Tractors is so
                                    dumb!'
```

- If you tip some of the tractors and make it all the way to the end, output the following string:

```
      'You tipped <# tractors tipped> tractors out of <total # of
            tractors> tractors. That's funny right there!'
```

Finally, write a file with your modified map. The output filename should have `'_updated'` appended to the inputted filename (e.g. Input: `'map.txt'` → Output: `'map_updated.txt'`). Your file output **should not** have a new line at the end.

Example:

`map.txt`

```
1| .....F.........................................>>>>>.........
2| ...>.........................E.....
3| ...^>.....TT......<......TTT.........
4| ..................TTTT
5| ....^...........<M.........................
6| .......TTTTT.....................TT........TTT............
```

```
>> str = tractorTipping('map.txt');

str → 'You tipped 7 tractors before being caught by Frank. You'd
better run or he's gonna get you!'
```

map_updated.txt

```
1| .....F.....................................>>>>>.........
2| ...##C.....................E.....
3| ...##################......TTT.........
4| ....#..............TTTT
5| ....#############.#.........................
6| .......TTTTT.............................TT........TTT............
```

Notes:

- The lines of the text file are not guaranteed to be the same length.
- Frank does not move.
- Tractors cannot be tipped more than once; in other words, **do not "double-count" tractors** that you encounter in your VF more than once.
- Any tractors that are in your VF at that moment you are caught are not counted (Do not check your VF if you are in Frank's VF).
- Do not count any new tractors that come into your line of vision when you land on the `'E'`
- The `'E'` will never be in the VF of Frank
- The current position tile is not part of the VF; i.e. if you encounter a `'T'` **at your current position**, do not tip it, as it is not in your VF. (Tractors directly on your path will be tipped anyways, as they will pass through your VF and will instead be counted from your VF check before leaving it and being replaced by a `'#'`)
- You might cross over previously travelled paths, but you will never go out of bounds of the file.
- Do not replace the final location `'E'` with a `'#'`, but you should replace the initial location `'M'` with a `'#'`.
- Only change tiles that appear directly in your path; you must replace these with a `'#'` in your file output. If they are only in your VF, they should not change, as you have not traversed on top of it.

Hints:

- Helper functions might be useful.
- You should store the data from the .txt file into another class type so that it is easier to manipulate the data.
- You may want to pad the map so you do not index out of bounds.
- Think about how you can use a mask to prevent double counting tractors