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.

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.

Also note: you can start the File Comparison Tool by clicking on "Compare" in the Editor ribbon if that is easier for you.

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.

Finally, please remember to **close all files that you open**. The use of `fclose('all')`, `fclose all`, or `close all` is not permitted, so make sure to close each file individually. 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.

Coding!
~Homework Team

Function Name: `functionHeader`

Inputs:

1. (*char*) A function name
2. (*double*) The number of inputs
3. (*double*) The number of outputs

Outputs:

None

File Outputs:

1. A .txt file containing the function header

Background:

You have been selected to be CS 1371 TA for a day! You take a break from video games to fulfill your duties as an honorary TA. You are writing a new auto-grader and have been tasked with making sure that students who use banned functions in their homework get a zero. Your idea is to overwrite each banned function with a "dummy" function that does not give the proper output. For example, if the `mean()` function is banned, your code should write a new `mean()` function that overwrites the built in one, so that if a student's code calls the `mean()` function, it will result in an incorrect answer.

Function Description:

Write a function that takes in a function name, a number of inputs, and a number of outputs, and writes a text (.txt) file containing the function header. In the function header, the base word for the variables should be in and out, respectively. After the function header, include a blank line, followed by the end keyword. As is the case with any MATLAB function, the file name should be identical to the function name.

Example:

```
>> functionHeader('myFunc', 3, 2)
myFunc.txt →
```

```
function [out1 out2] = myFunc(in1, in2, in3)

end
```

Notes:

- If there are no outputs, you should include empty brackets and the assignment operator.
- Do not put commas between outputs.

Continued...

- Be careful with spaces; your file should match the solution file exactly. If you are unsure, run the solution file and use `visdiff()` to compare your output file to the solutions.
- There should not be an additional newline at the end of the file.

Function Name: `aceAttorney`

Inputs:

1. (*char*) Filename of the speech to analyze, with extension

Outputs:

1. *(char*) A formatted sentence announcing the score

Background:

You are Phoenix Wright, a rookie lawyer working your first case, and it's a big one! You've heard testimony from some witnesses, and you need to establish their credibility before the court. You excelled at your MATLAB classes in law school, so you decide to have MATLAB help you analyze this testimony!

Function Description:

Write a function in MATLAB that analyzes a text file containing somebody's testimony and outputs a score and sentence announcing the reliability score. The text file's first line will always be formatted like this: `<Title>-<Author>`. Each testimony starts at a score of 100. To calculate the score, use the following rules:

- Subtract 3 points for every `'uh'`
- Subtract 1 point for every `'like'`
- Add 10 points for every `'THWG'`

The final output sentence should read:

```
'<Author>'s <Title> got a speech score of <score>!'
```

Example:

If `ex.txt` file contained the following:

```
Example-MATLAB
This is uh an example.
Speeches like this are like, uh, bad.
```

```
>> sentence = aceAttorney('ex.txt')

sentence → 'MATLAB's Example got a speech score of 92!'
```

Notes:

- When checking for `'uh'`, `'like'`, and `'THWG'`, case does not matter.
- Words will be separated by spaces, but you can ignore punctuation in the line.
- Do not count `'uh'` or `'like'` if they appear inside another word (i.e. `'likes'`)

Function Name: `marioKart`

Inputs:

1. (*char*) The filename of the game records

Outputs:

None

File Outputs:

1. A .txt file containing the winner and their score

Background:

You and your friends are playing Mario Kart for the twentieth time today and want to find out who got the most points overall. You set out to create a MATLAB function to take the number of wins each player earns and assign points to determine the true winner.

Function Description:

Write a function in MATLAB to determine the winner based on how many times the players placed in the top three. Every 1st place earns 15 points, every 2nd place earns 12 points, and every 3rd place earns 10 points. Create a file named `'<filename>_winner.txt'` containing the name of the winning character and the number of points they earned.

Each line of the input file will have this format, where `num` is the number of races in which `name` finished in `place` :

```
'Name: place:num place:num place:num'
```

Example:

`Score.txt` →

```
Mario: 1st:3 2nd:3 3rd:14
Luigi: 2nd:14 1st:2 3rd:4
Princess_Peach: 1st:15 3rd:2 2nd:3
```

```
>> function marioKart ('Score.txt')
```

`Score_winner.txt` →

```
Winner: Princess_Peach
Points: 281
```

Notes:

- There will always be a space between the name and the places.
- The places aren't guaranteed to be in the correct order.
- There will be no ties.

Hints:

- Think about how you stored the maximum value for minMax.

Function Name: `civLeaders`

Inputs:

1. (char) Name of a text file with instructions

Outputs:

None

File Outputs:

1. A .txt file with the completed adventure

Background:

You've decided to play a game of Civilization IV (the best Civilization game*)! The first part of playing a game of Civilization is picking which leaders will be in the game. Each civilization has several leaders to choose from, and it's hard for you to decide, so you turn to MATLAB to help you choose!

Function Description:

You will be provided a text file that contains which leaders you pick from each civilization. Each line will be of the format `'civilization.txt:leader'`. Each civilization file will be separated into the leaders you can pick, where each leader has a description paragraph. In each file, the first line will be the name of one of the leaders, followed by a colon. The paragraph associated with that leader will follow. Eventually, another leader's paragraph will occur, again with a colon, and then the paragraph related to that leader.

For each line in the input file, open the text file specified to the left of the colon. Find the leader whose name is to the right in that text file, and then copy the leader name and the associated paragraph into a new text file with '_leaders' appended to the name of the input text file.

Notes:

- Each leader's name will be one word.
- The descriptions will not contain colons
- There should be no extra new lines between paragraphs or at the end in the output file.

* According to one TA

Continued...

Example:

cs1371.txt →

```
kantwon:
Kantwon Rogers is a graduate student in the Human-Computer Interaction degree program.
He also earned a BS in Computer Engineering and a MS in Electrical and Computer Engineering from Georgia Tech.
His research interests lie in the area of computer science education and increasing the involvement and success of Black students in the field.
smith:
David Smith is and Aerospace Engineer with 31 years of experience in industry.
He is in his 20th year as a Lecturer in the College of Computing.
His primary accomplishment has been to design and implement CS1371.
```

civ1.txt →

```
cs1371.txt:kantwon
```

`civLeaders('civ1.txt')`

civ1_leaders.txt →

```
kantwon:
Kantwon Rogers is a graduate student in the Human-Computer Interaction degree program.
He also earned a BS in Computer Engineering and a MS in Electrical and Computer Engineering from Georgia Tech.
His research interests lie in the area of computer science education and increasing the involvement and success of Black students in the field.
```

Function Name: `evilIsAfoot`

Inputs:

1. (*char*) Name of a text file, with extension
2. (*char*) Sequence of movement commands

Outputs:

1. *(char*) A string of the characters covered by the cursor

Background:

I don't get paid enough for this, you think to yourself for the third time that night. Private eyes rarely do.

A week before, you'd been lounging in the chair behind the desk of your office on the third floor of a dingy New York apartment building. Simpler times, those. Sure, there wasn't enough money for dinner that night, but you would've survived.

The prospect of an unusually well-paying and ostensibly straightforward infidelity case was too tempting to pass up, which is perhaps why you chose to overlook the strange behavior of the trench-coat wearing figure who knocked on your door that night.

At least this case was turning out to be quite interesting for a cheating job. Unfortunately, you were quickly reaching the point of being in over your head. What had initially promised to be an uneventful night tailing your client's spouse on a Friday night ended up leading you down a rabbit hole of white-collar crime, blackmail, and, most recently, corporate espionage.

The backdoor of the Evil Corp™ headquarters had been surprisingly easy to get through with little more than a borrowed employee badge and some confidence. All you had to do now was find the right computer, get what you came for, and get out before security had a chance to notice you.

It doesn't take long to find the computer you're looking for. Of course, you realize, it's not going to be as easy as a drag-and-drop. It's a server computer, so all you have to work with is a keyboard and a terminal whose text editor hasn't been updated since the early 90s. Your terminal skills are a bit rusty, but now's not the time for a hacking montage, and you can hear the soft rhythm of distant footsteps elsewhere in the otherwise quiet office building. It's time to get started.

Function Description:

VIM (originally vi) is a console-based text editor designed for computers before mice and arrow keys on the keyboard were widespread. As a result, movement around a file was done with the `h`, `j`, `k`, and `l` keys, which move the cursor in four directions around the screen. Your job is to write a function to take a text file and a sequence of these movements and determine which characters the cursor hovered over during the sequence.

Continued...

You will be given a text file and a sequence containing some combination of the following movement keys:

Key	Direction
`h`	left
`j`	down
`k`	up
`l`	right

Write a function to follow the sequence of movements through the text file and output a string containing the character hovered over by your cursor after each individual movement.

Example:

In the following example, space characters are represented as center dots (`·`) to make movements through spaces easier to see. The highlighted characters are characters visited by the sequence, and the red highlighted character is the final character visited.

```
sleeplessCity.txt ⇒

Out·in·the·sky,·no·one·sleeps.·No·one,·no·one.····························
No·one·sleeps.············································································
The·creatures·of·the·moon·sniff·and·prowl·about·their·cabins.·············
The·living·iguanas·will·come·and·bite·the·men·who·do·not·dream,···········
and·the·brokenhearted·fugitive·will·meet·on·street·corners················
an·unbelievable·alligator·resting·beneath·the·tender·protest·of·the·stars.
··························································································
Out·in·the·sky,·no·one·sleeps.·No·one,·no·one.····························
No·one·sleeps.············································································
In·a·graveyard·far·off·there·is·a·corpse··································
who·has·moaned·for·three·years············································
because·of·an·arid·landscape·in·his·knee;·································
and·that·boy·they·buried·this·morning·cried·so·much·······················
it·was·necessary·to·call·out·the·dogs·to·keep·him·quiet.··················

sequence ⇒ 'jjllllkhjjjjkjlllljjjjjjkjjlll'

evilIsAfoot('sleeplessCity.txt', sequence) ⇒ 'ONThe·cno···unbel·tsa··tnece'
```

Continued...

Notes:

- The cursor always starts on the first character in the file.
- Every line in the text file will be the same length. Lines with fewer letters will be filled with spaces as in the example above.
- Your output should begin with the character your cursor starts on and end with the character you end on, and its length should be the length of the input sequence + 1.
- The sequence will never take you out of bounds of the file.

Hints:

- Don't be afraid to close and reopen the file as many times as you want.