Drill Problem #1

Function Name: detention

Inputs:

1. (*char*) A string of the name of a .txt file
2. (*char*) A string of the exact sentence that needs to be written

Outputs:

1. (*char*) A string of a sentence indicating whether each line of the .txt file matches the exact sentence from the second input

Function Description:

"Writing Lines" is a classic punishment for naughty students that wind up in detention at school. The teacher writes a line on the board, usually describing what the student did wrong, and the misbehaving student is expected to write this line over and over again a certain number of times until the student may be dismissed.

Your task is to write a MATLAB function that can check all the lines, which are recorded in a ".txt" file, that the student turns in and determine whether the student wrote every single line properly or if they messed up their punishment and need to repeat it.

You will need to iterate through every single line of the input ".txt" file and compare each line to the exact sentence given as the second input. The output of your function should be a sentence stored in a string that indicates the results.

If every single line matches the second input exactly, your result should read:

```
'Good job. You are free to go!'
```

If one or more of the lines does not match the second input exactly, your result should read:

```
'Not so fast. Start all over!'
```

Notes:

- The number of lines contained inside the ".txt" file is irrelevant.
- Your string output must match the solution file exactly in order to receive credit.

Hints:

- You may find the `fgetl()` and `all()` functions useful.

Drill Problem #2

Function Name: hwGrader

Inputs:

1. (*char*) The name of a `'.txt'` file containing the correct outputs for the test cases
2. (*char*) The name of a `'.txt'` file containing a student's outputs for each test case

Outputs:

1. (*double*) A percentage number reflecting the student's score on the problem

Function Description:

Congratulations, you have just been hired as a CS 1371 TA! Your first job is to write a new auto-grader for the homework problems. This auto-grader should be a MATLAB function called `hwGrader()` that outputs a student's homework percentage based on two given text files. The first text file is the solution file containing the correct answers to test cases, and the second text file is the student's answers. The files are guaranteed to be formatted in the following way:

```
function name: <name>
<variableName1> = <value>
...
<variableNameN> = <value>
```

Where the first line gives the name of the function, and each of the following lines is a test case with the variable name separated from its value by an equal sign.

In order to grade the student's code, you must first check to see if he or she named the function correctly. If the first line in the student's text file does not match the first line in the solution's text file, then the score should be `0`.

If the first lines do match, then the score should be the percentage (rounded to an integer within the range 0-100) of test cases that the student got correct. A test case is considered correct if the `'<value>'` in the student case and the test case is exactly the same. Thus, if 1 out of the remaining 3 lines (not counting the first line) in the text files match, then the student should receive a score of `33`.

Notes:

- Your input file names will already contain the `'.txt'` suffix.
- It is guaranteed that there will be no empty lines in the text files, i.e. if a file has 4 total lines, then it must have the function name line and 3 test cases.
- The two input files are guaranteed to have the same number of lines.

Drill Problem #3

Function Name: krustyKrab

Inputs:

1. (*char*) The name of a '.txt' file indicating the check from an order at the Krusty Krab

Outputs:

1. (*double*) The total for the Krusty Krab order, including a 15% sales tax

Output Files:

1. A '.txt' file that contains the full receipt of the order from the Krusty Krab

Function Description:

Are ya ready Kids? Aye-aye, captain! I can't hear you!! AYE-AYE CAPTAIN!! Ohhhhh… Welcome to Bikini Bottom! Home of the cartoon favorite SpongeBob SquarePants. SpongeBob works as a fry-cook at the town's favorite undersea eatery: the Krusty Krab. The Krusty Krab is owned by the stingy Eugene Krabs, who is looking for someone to help him keep track of all his orders and more importantly…keep track of all his money!

Write a MATLAB function called `krustyKrab()` that takes in the name of a '.txt' file which has a list of what a particular customer ordered. The first line of the text file will contain the customer's name. Each additional line of the text file will have the item the customer requested and its standard price from the menu. The `krustyKrab()` function needs to output the total of the order, including a 15% sales tax (because Krabs likes his money!) as well as a new '.txt' file that has the full receipt from the order.

The receipt (your output text file) will need to include the exact same lines that were printed on the original '.txt' file, plus the following three lines:

1. The subtotal of the order (the sum of all the prices of each item ordered)
2. The total of the order rounded to two decimals (the subtotal with a 15% sales tax)
3. The Krusty Krab slogan: `'The Krusty Krab, Come spend your money here!'`

The name of the output '.txt' file should be the name of the input '.txt' file given with '_receipt' appended to it.

For example, given the following input text file named `'Order1.txt'`:

```
1 Name: Patrick Star
2 Krabby Patty        $2.00
3 Coral Bits          $1.95
4 Small Sea Foam Soda $1.00
```

This function should output the total cost from the receipt, including 15% sales tax, which is rounded to two decimals, as a double:

```
total => 5.69
```

The output file will be named `'Order1_receipt.txt'` and have the three new lines in addition to the lines from the original text file.

```
1 Name: Patrick Star
2 Krabby Patty        $2.00
3 Coral Bits          $1.95
4 Small Sea Foam Soda $1.00
5 Subtotal=$4.95
6 Total=$5.69
7 The Krusty Krab, Come spend your money here!
```

Notes:

- The lines indicating which items were ordered should appear EXACTLY the same in the receipt '.txt' file you output as the input file, including the customer's name being first.
- The subtotal and total lines should be formatted EXACTLY as indicated, i.e. the word, an equal sign, a dollar sign, and then the value.
- There should NOT be an extraneous empty line at the end of your output '.txt' file.

Hints:

- The prices of each item in the given '.txt' file will always appear immediately after the '$' character present in that line.
- When using `fprintf()`, in order to format your decimals correctly for the subtotal and total values, you must use %0.2f as your variable in the string. Therefore, the line `fprintf(fh2, 'Subtotal=%0.2f', 5);` prints the following line to the specified file >> `Subtotal=$5.00`, where, the 'f' stands for floating point number, and will avoid any weird display of your rounded number. The '0.2' indicates that it will only display two places after the decimal point.

Drill Problem #4

Function Name: australianSecretAgent

Inputs:

1. (*char*) A string containing the name of a conversation saved as a text file

Outputs:

1. (*char*) The name of the Australian secret agent

Function Description:

You've begun your first week at the Central Intelligence Agency, but all is not well. It has recently come to light that our ally, Australia, has planted a mole in the CIA to discover the most heavily guarded secret in US history: why kids love the taste of Cinnamon Toast Crunch. Your assignment is to find the mole.

Luckily for you, you've just taken CS 1371 and the CIA keeps all conversations between its employees saved as plain-text documents. In your infinite wisdom you've set out to find the mole using a MATLAB function. You know that Australians are notoriously bad at hiding their (fantastic) cultural shorthand, namely their use of "prawn", "barbie", and "ripper" when discussing shrimp, barbeques, and things that are awesome, respectively.

Write a function that will look for the strings: `'prawn'`, `'barbie'`, and `'ripper'` within each person's portion of the dialogue. If there are more than 4 total occurrences of any of the patterns in either person's speech, they are certain to be the secret agent and your output should be their name. There will not be any ties.

The conversations will be formatted as such:

```
Phillip: Hello there Charles!
Charles: Why hello there Phillip! Isn't the weather ripper?
Phillip: Charles, are you Australian?
Charles: Why yes I am! How did you know? Was it my constant mention of
prawns, barbies, and my description of things as being "ripper"?
Phillip: It may very well have been!
```

Notice that each line (the fourth merely runs out of space, there is no new-line character) alternates between the two members of the conversation, and that it is formatted as

`['Name',':',' Speech']`

Notes:

- The pattern may show up as a part of a larger word. For example, `'barbie-doll'` would count as an instance of `'barbie'`.
- The patterns are NOT case-sensitive. For example, `'Barbie'`, `'BaRbIe'`, and `'barbie'` would all count as instances of `'barbie'`.

Hints:

- The function `strfind()` is your friend. Don't neglect your friends.

Drill Problem #5

Function Name: ottendorf

Inputs:

1. (*char*) The beginning of the filenames of the '.txt' files being used as reference passages
2. (*char*) The name of a '.txt' file containing the cipher

Outputs:

1. (*char*) The secret message decoded using the cipher and reference passages

Function Description:

An Ottendorf—or book—cipher is a type of encoding method (referenced in the movie National Treasure) in which sets of numbers correspond to letters in some set of reference materials. In this variation, each letter will be encoded in the following manner:

```
<reference #>-<line #>-<word #>-<letter #>
```

For example, the cipher line 3-23-8-4 corresponds to the fourth letter of the eighth word in the twenty-third line of the third reference text file. The first number would usually represent a page number in a book, so you can think of each reference file as a single page of a whole unified reference.

Write a function that takes a file of encoded letters (one letter's sequence per line) and outputs the message decoded using a given set of reference documents.

Notes:

- The reference filenames will always be formatted as `'<input1>_#.txt'`. For example, if the first input of the function were `'hpLastPage'` and the current cipher line were 3-23-8-4, you would be looking in the file called `'hpLastPage_3.txt'`.
- The end of the cipher file will be marked with a line containing 0-0-0-0
- Blank (empty) lines in the **cipher** should be represented as spaces in the decoded message output
- Lines of punctuation in the **cipher** should be carried over to the output string as well.
- You do not need to worry about finding blank lines or punctuation in the reference files; no words with quotes, hyphens, etc. will be encoded.

Hints:

- Do not forget to close files as you finish using them.
- You should only tokenize using spaces.
- Remember that lines read in from a file are strings (the `str2num()` function may be useful).
- Also remember that `str2num()` will yield an empty vector if the string does not contain a number.

Drill Problem: EXTRA CREDIT

- THIS PROBLEM IS EXTRA CREDIT! -

Function Name: rhymeTarragon

Inputs:

1. (*char*) The name of a '.txt' file containing a possible poem

Outputs:

1. (*char*) An output string describing whether the text in the file is a poem

Function Description:

'You're a poet,
And you know it,
And you're about to show it!'
-CS 1371 TA's

Remember the `rhymeThyme()` function from Homework 4? It's back! Now that you know how to determine whether two sentences rhyme, you will extend rhyming from two sentences to a collection of lines and determine whether these lines form a poem.

Write a function in MATLAB called `rhymeTarragon()` that takes in a file as an input and outputs a description of whether the file is a poem or not. You will do this by determining the rhyme scheme of the poem in the file. A rhyme scheme is a way of succinctly showing which lines in a poem rhyme together. The first line in a poem is assigned 'A', and every line that rhymes with that line is also assigned 'A'. The first line that doesn't rhyme with the 'A' line is assigned 'B', and all lines rhyming with that line are assigned a 'B', and so on through the alphabet as far as it needs to go. For example, the poem:

Tests are hard,
But you'll do well,
Coding ain't easy,
But MATLAB is swell!

has a rhyme scheme 'ABCB'. Given a text file containing a poem, determine the rhyme scheme of the poem by following the same rhyming conventions as the `rhymeThyme()` function; use the following table to determine whether the file is a poem:

Rhyme Scheme	Type
`'ABAB'` , etc.	Alternating Rhyme
`'ABBA'` , etc.	Enclosed Rhyme
`'ABCB'`	Ghazal
`'AABBA'`	Limerick
`'AABA'` , etc.	Ruba'i
`'ABBAABBACDECDE'`	Petrarchan Sonnet
`'ABABCDCDEFEFGG'`	Shakespearean Sonnet

You will then output a string describing the rhyme scheme and type in the format:

```
The file contains a <type> with an '<rhyme scheme>' rhyme scheme.
The file contains an <type> with an '<rhyme scheme>' rhyme scheme.
```

Where you replace <type> and <rhyme scheme> with the poem type and rhyme scheme, respectively (please note the strings above appear as they would appear in the Command Window). If <type> starts with a vowel, the second output (with an `'an'`) should be used, if <type> starts with a consonant, the first output (with an `'a'`) should be used. Note that there are single quotes surrounding <rhymeScheme> in the outputs. This means that if the file contains a poem with rhyme scheme 'ABCB', the correct output is:

```
The file contains a Ghazal with an 'ABCB' rhyme scheme.
```

If the rhyme scheme does not match any of the listed rhyme schemes in the chart, output the string:

```
The file does not contain a poem.
```

Finally, 3 of the rhyme schemes in the chart above are followed by 'etc.' This means that if the 4 letter pattern continues deeper into the alphabet, it still is considered that type; however, it must continue deeper into the alphabet, not simply repeat. The rhyme scheme 'ABABABAB' will not be considered a poem, but 'ABABCDCD' is considered a poem with alternating rhyme. The output for that second case should be:

```
The file contains an Alternating Rhyme with an 'ABABCDCD' rhyme scheme.
```

This is only the case for those 3 cases followed by 'etc.' – the others must match exactly to be considered a poem.

Notes:

- There will be no lines in the file without text (no blank lines).
- All lines will have at least 3 characters.
- The words in each line are separated by a single space.
- There will be a comma at the end of every line, except the last line, which will end with a period.
- The rhyme scheme will never go further into the alphabet than 'G'.
- You must use a modified `rhymeThyme()` function as a helper function; you may have to make a change in the code—namely, the `rhymeThyme()` function takes in two sentences separated by a period, and the lines in the file will end in commas.

Hints:

- Your `rhymeThyme()` helper function will work with 2 sentences together in a single string—think about how you can combine two strings into a single line to make use of the `rhymeThyme()` function.
- To make a single quote appear in a string, you must use back to back single quotes (`''`).