You've made it to your very last CS 1371 homework assignment! This homework is completely optional, but there are several incentives for doing it:

1. It is worth 1 homework grade of extra credit.
2. It is designed to cover most of the topics covered throughout the course to refresh your memory before the final.
3. There is a surprise waiting for you at the end if you complete the whole thing!

Thanks for a great semester and good luck on this and all other finals! And as always…

Happy coding,
~Homework Team

Function Name: aliBabaAndTheFortyThieves

Inputs:

1. *(char)* The name of an Excel file

Outputs:

1. *(struct)* A structure array with the data from the Excel file

Banned Functions:

`cell2struct()`

Function Description:

After defeating the Forty Thieves, Ali Baba decided to explore the secret cave and see what riches were stored there. During his search, he came across a secluded alcove in the back, with forty little cubbyholes. In each one was a flash drive with a single, mysterious excel file on it. Ali Baba was baffled by this discovery, since at this time, Bill Gates had not even been born and Microsoft did not exist. He decided to take these flash drives to the wisest person he knew; you, the town MATLAB specialist! Luckily, you let Ali Baba know that MATLAB is so amazing that it has a function called `xlsread()`, which is able to read an Excel file even though Excel files do not exist yet! Unfortunately, Ali Baba tells you that he has never seen a spreadsheet before and that he will not be able to interpret it, so he asks you to convert the Excel file into a format he is more comfortable with: a structure array. You happily oblige, since you love MATLAB, cell arrays, and structures!

The structure array should be 1x(M-1), where M is the number of rows in the excel file. The field names will always be in the first row of the Excel file. Each row of the Excel file corresponds to a single structure in the structure array. Each column of data corresponds with a field named in the first row of the Excel sheet. However, you are not guaranteed that every column in the first row will contain a field name; some may be blank. If a column does not contain a header in the first row, then data in that column belongs to the preceding field (whichever column to the left that does have a header). For each non-empty row in a header-less column, you should nest the data within a cell array with the data to its left and right, starting at the column with the previous field until the column preceding the next field. If a data value does not exist for a column with a header, the corresponding field should be empty. If a data value does not exist for a column without a header, then you should not include it in your cell array. An example is shown on the following page so you can visualize this.

```
ExcelFile =>
```

'Name'	'Foods'			'Color'	'Age'	'Style'	
'Steve'	'Salad'			'Purple'	19	'Modern'	'Classic'
'Ashley'	'Cake'	'Candy'	'Pie'		21	'Hipster'	'Regal'

```
sa(1) =                          sa(2) =
Name: 'Steve'                    Name: 'Ashley'
Foods: {'Salad'}                 Foods: {'Cake', 'Candy', 'Pie'}
Color: 'Purple'                  Color: []
Age: 19                          Age: 21
Style: {'Modern', 'Classic'}     Style: {'Hipster', 'Regal'}
```

Notes:

- Note that the length of the cell arrays is not always the same for the same field in different structures.
- A field name will only appear in the header row once. All field names will be unique.
- The only data types you will encounter in the excel file are double and char. Therefore, the cell arrays can only contain doubles and/or chars.
- The data type for a given column in the Excel file, excluding the header, will be homogeneous.
- If a cell in the Excel file is empty and the next column has an empty header, the cell in that next column and same row will also be empty.

Hints:

- Be careful analyzing the structure output; the way MATLAB displays data can trick you about the class of that data.
- http://cs1371.gatech.edu/pokedex/

Function Name: clownTown

Inputs:

1. *(struct)* A 1xN structure array containing doubles, chars, and cell arrays

Outputs:

1. *(char)* A string that will be the first passcode
2. *(double)* A number that will be the second passcode

Function Description:

Congratulations! You have just been elected as the mayor of Clown Town, and will soon take on all the responsibilities that come with being the leader of the performing world. One such responsibility is possessing all of Clown Town's nuclear codes (clowns can be scary too). There are two separate passcodes required to access these codes, and because you are a new mayor, you have been asked to create new passcodes for your term. The first passcode will be a string, and the second will be a number.

Because you are a mayor of the people, you want to create your passcodes using the personal information of all of the residents in Clown Town. Lucky for you, all of this information is stored in City Hall as a MATLAB structure array, and even more fortunately, you took CS 1371! Here's what you have to do.

To create the string that will be your first passcode, you will take out all the field names from the structure array, order them alphabetically, and then draw a single character from each field name. The character that you select will be in the *i*-th position of the fieldname, where *i* is the total number of words contained within that field. If the number of words exceeds the number of characters in the field name, then your indexing should wrap around. For example, if a field name has five characters, but you count seven words in that field, then you should index out the second character. If there are no words in that field, then you should use a space for that field name.

As you are counting words, you should know that the structure array can contain three different data types: double, char, and/or cell. Any doubles can be ignored, as these are not words. You should consider all spaces in a string to delimit a word. All cell arrays will be 1xM, and each cell will contain either a double or a char, and these can be counted in the same fashion that was just stated.

As for your second passcode, the number will be the mean of all of the numbers contained in the structure array, rounded to the nearest whole number. If there are no numbers, then make this value 0. An example is given on the following page.

```
sa =>
sa(1) =                        sa(2) =

LastName: 'McDonald'           LastName: 'the Clown'
FirstName: 'Ronald'            FirstName: 'Krusty'
Age: 53                        Age: 'Fifty-two'
Address: '45 Every Corner'     Address: '123 Springfield Lane'
Occupation: 'clown'            Occupation: 'clown'
Ch1ldr3n: []                   Ch1ldr3n: {3,'Bart','Lisa','Maggie'}
AnnualSalary: 18,870           AnnualSalary: 0


first passcode => 'sA 1isc'
second passcode => 4732
```

Notes:

- In the example, observe what happens for Address field. Because `'45 Every Corner'` is of type char, it counts as three words.
- In the example, second passcode is the mean of 53, 3, 18870, and 0.
- When sorting the field names, just use MATLAB's method of sorting. Do not worry about distinguishing between sorting lower and uppercase letters.
- You are not guaranteed any patterns. For example, there is no guarantee that a field will only contain all doubles or all chars.
- There will never be extraneous spaces.
- Cell arrays can be of different lengths, but will always be one-dimensional.
- You will never have a nested cell array.

Hints:

- `mod()`

Function Name: onceUponAPage

Inputs:

1. *(char)* A string to be converted to a file name
2. *(double)* The line number to search up to

Outputs:

1. *(double)* A 1x5 vector representing the number times O, D, L, A, and W appear in the text file, in that order

Function Description:

Tired of studying for finals, you and your roommates unwind by playing game. You write each other cryptic notes, which lead you to a text file containing a page of a book. Then, you must search through the text file up to a given line number, counting the number of times you come across the letters O, D, L, A, and W. Store the number of occurrences in a vector in the given order.

Convert the string to a file name using the following process:

1. Remove all vowels and non-letter characters from the string.
2. Capitalize any W's in the string.
3. Make all other letters lowercase.
4. Add `'.txt'` to the end of the string.

Notes:

- You should count both uppercase and lowercase occurrences of the letters.
- Search all lines UP TO, not including, the line indicated by the line number.
- If the given line number is greater than the lines in the file, read to the end of the file.

Function Name: theNastyNasties

Inputs:

1. *(double)* A 1x5 vector
2. *(double)* An x-bound.

Outputs:

1. *(double)* A 1x50 vector representing x-values
2. *(double)* A 1x50 vector representing y-values

Function Description:

Fun puzzles don't always have fun names, like this function you're about to write! Given a 1x5 vector of doubles, find the arc length bound by zero and the x-bound of the curve given by the quartic (4th degree) polynomial whose coefficients are the elements of the vector in the first input. The arc length is given by the following expression:

$$L = \int_a^b \sqrt{1 + [f'(x)]^2}\,dx$$

For this function, you will be permitted to use the integral function and function handles: `integral(FUN, A, B)` where `FUN` is a function handle, `A` is the lower bound of the integral, and `B` is the upper bound of the integral. A function handle is a MATLAB variable that provides a
means of calling a function. For example, the function:

$$f(x) = cos(x) + x^2 + e^{3x}$$

can be represented by the function handle:

```
f = @(x) cos(x) + x.^2 + exp(3.*x)
```

Once you type this code in MATLAB, the variable `f` is now a "handle" to that function. So you can evaluate $f(3)$ by typing `f(3)`. A function handle like this can be used as the first input to the `integral` function. If you are interested, function handles are a very powerful tool in MATLAB and you can read more about them here.

Next, divide the arc length by the x-bound to find the average length of an arc segment over one unit along the x-axis. Iterate from zero to the x-bound in steps of 0.25 to determine the x- and y-values of the points between which the length of the arc segment defined by the two points is greater than or equal to the length of the average arc segment. Starting from (0, 0), you must round and then store the x- and y-values of the first point after (0,0) at which the arc defined by it and (0, 0) is greater than or equal to that of the average length of an arc segment.

The next pair of x and y-values to be added correspond to the next point where the arc length between it and the last defined point is greater than or equal to the length of the average arc segment, and so on. Only append the rounded x- and y-values to the corresponding vectors if both of them are not already present in the corresponding vectors. Store only the first 50 points after (0,0) that satisfy these conditions.

i.e. if we want to add in the point (23, -43)

```
x = [5 3 109 -3 23 420]; % 23 is present in the x vector and -43 is
y = [7 -43 8 10 79 23];  % present in the y vector, so we do not add
                         % the point

x = [5 -43 3 29 23 7]    % 23 is present in the x vector but -43 is
not   y = [7 23 88 -11 28 17]  % present in the y vector, so we add
the
                         % point (there will be a duplicate 23 in the
              x
                         % vector
```

Finally, you will need to make sure that neither of the vectors contain values less than 1, and that none of the values exceed the x bound. Any value less than 1 should be replaced with 1. To normalize the y-values, perform the following. First change any negative y-values to their absolute value. Find the maximum y-value of the vector, and determine, for each y-value, the ratio of the y-value to the maximum value, and multiply this ratio by the x-bound to form new y-values. Then, round these modified y-values to the nearest integer.

This was a lot of detail, but here's a TL;DR: Create a vector of x and y coordinates where adjacent points represent a range over which the arc length is greater than or equal to the average arc length over the total range. Do not add duplicate points. Then perform some normalization on the x and y values as specified above.

Notes:

- `help integral`
- `help function_handle`
- The first point in the output vector, (x_1, y_1) indicates that the arc length between $(0, 0)$ and (x_1, y_1) is greater than or equal to the average arc length. However, the point $(0, 0)$ should never be included in the output vector.

Function Name: theGreatPortraitExhibition

Inputs:

1. *(double)* A 1x50 vector representing rows
2. *(double)* A 1x50 vector representing starting column values
3. *(char)* The name of the image file in which your portrait is hidden

File Outputs:

1. The image file of the portrait you found

Function Description:

Encryption comes in all shapes and sizes, including in graphics and visuals! In this problem, you will use a vector of row indices and a vector of column indices to extract a 50x50 portrait image from the larger input image file, then follow a set of predetermined manipulations to restore the portrait to its normal, readable, self. The rows of the portrait image are dispersed throughout the large image file at the row index from the first input and starting at the corresponding column index. For example, if the first values in the row vector and column vector are 16 and 72, respectively, then the first row of the portrait will be in the large input file starting at row 16, column 72, and continuing in that row for the width of the portrait (50 pixels).

Once you have pulled out the pixels in the final image, you should "undo" the following manipulations to produce the final image.

- The red layer of the portrait has been negated and transposed.
- The green layer of the portrait has it's 1st and 3rd, and 2nd and 4th, quadrants swapped (in which the first quadrants is the top left, second quadrants is the top right, third quadrant is the bottom left, and the fourth quadrant is the bottom right).
- The blue layer of the portrait has been rotated 180 degrees.
- Finally, the layers (red, green, blue) have been concatenated in the opposite order. This means that in the above description, the red layer, for example, is actually the third layer in the original image.

Once you have decoded the portrait from the original file, output this new image as a file. Name this new file the same as the input file name, except that instead of having `'_hidden'` before the extension, put `'_decoded'`.

Notes:

- All output images should be saved as .png regardless of input image type.

Congratulations, you have finished all of the homework for CS 1371! To reward you for your efforts, we have hidden a secret output in this homework that can only be found by chaining all of the functions for this homework together. Once you are finished with everything, try running the following block of code:

```
[a, b] = clownTown(aliBabaAndTheFortyThieves('soExtra.xlsx'));
[x, y] = theNastyNasties(onceUponAPage(a, b), 400);
theGreatPortraitExhibition(x, y, 'theGPE_hidden.png');
imshow('theGPE_decoded.png');
```