Function Name: `structDisp`

Inputs:

1. (*struct*) An MxN structure array

Outputs:

1. (*char*) A PxQ character array where P = (number of fields + 1) * M and Q = 50 * N

Function Description:

MATLAB does fine when displaying a single structure. It shows all of the fields and their contents. But when you try to display a structure array, it just says, "MxN structure array with the fields:" and then lists the fields. This is not particularly useful if you want to see the data contained in the structure array. So you are going to write a function to remedy this problem!

You will output a character array that shows the entire structure array, including its contents. The structure array will be a concatenation of the output that MATLAB returns for an individual structure. You have been given a helper function `struct2charArr()`, which takes in a 1x1 structure and turns it into an MxN character array, where M is the number of fields in that structure plus one (there is an extra empty line at the bottom meant to create space between this and the next structure), and N is the minimum number of columns needed to fully capture the data in the fields. So if there is a long vector contained in one of the fields, N may be very large. On the other hand, if all of the fields in the structure contain short data entries, N may be small. The easiest way to see what the helper function outputs is to try it with different 1x1 structures.

Given this unwanted variation in column sizes, you should manipulate each output character array so that it contains exactly 50 columns. In other words, if the data in one structure is short, you will need to extend (the right side of) the columns of the output of the helper function with spaces. If the data is long, you will need to remove some of the output. Simply remove all columns after 50 and don't worry about displaying the overflow data.

Finally, you will concatenate each character array into your larger output character array in the dimensional order that the structures appeared in the original input. This concatenation is similar to the `layerBricks()` problem you did in Homework 07.

Notes:

- Use the `isequal()` function with your code and the solution code to check your answers. You should be doing this anyway, but it is especially important on this problem as there are lots of spaces in these character arrays and you won't be able to easily see differences.
- The size of a single character array output from the helper function will have the rows dependent on the number of fields, but remember that the size of a structure is independent of the number of its fields.

Function Name: `structFind`

File Inputs:

1. (*struct*) A 1xN structure array
2. (*char*) A string to search for

Outputs:

1. (*cell*) The index and fieldname under which the string can be found

Function Description:

Last week you wrote a function called `cellSearch` that would return the value at a given list of indices within a cell array. This week, you will basically be writing a function to perform the opposite function on a structure array.

The function will take in a structure array and a string to search for within the structure array. If the string is contained anywhere in the structure array, the function will output a 1x2 cell array with the first element being the numerical index of which structure contains the string and the second element being the specific field name that contains the string.

In other words, if the inputs to this function were `stArr` and `'example string'` and the function produced the output: `{[5], 'banana'}`, then typing:

```
>> stArr(5).banana
```

in the Command Window could return something like: `'here is an example string'`

If the string does not exist in the structure array, the function should return a 0x0 empty cell, `{}`.

The input structure is guaranteed not to have any nested structures, but could be of any length and contain any number of fields. If the string occurs multiple times in the structure, this function should return the location of the **first** occurrence. Also note, that the string to search for could be a substring of a string in the structure.

Function Name: `colorGrad`

Inputs:

1. (*struct*) An MxN structure array with red, green and blue intensity values
2. (*char*) The color to determine sorting in the row direction (`'red'`, `'green'`, or `'blue'`)
3. (*char*) The color to determine sorting in the column direction (`'red'`, `'green'`, or `'blue'`)

Outputs:

1. (*struct*) The sorted structure array

Function Background:

The RGB color model is commonly used to display images on televisions and computers. It is an additive model in which red, green and blue are combined in varying intensities to create a broad spectrum of colors. You will learn more about images in a few weeks, but for now we will implement them using structures.

Function Description:

Your first input variable will be an MxN structure array with three fields. Each of the fields will contain an 'intensity value' for the colors red, green and blue respectively. The fields will always be named 'red', 'green', and 'blue' but may be in any order or have varying case. For example, the fields 'Green', 'RED', and 'blue' would also be valid. Given this structure, the function should:

1) Sort each row of the structure array using the first color's intensities (from least to greatest)
2) Sort each column of the structure array using the second color's intensities (from least to greatest)

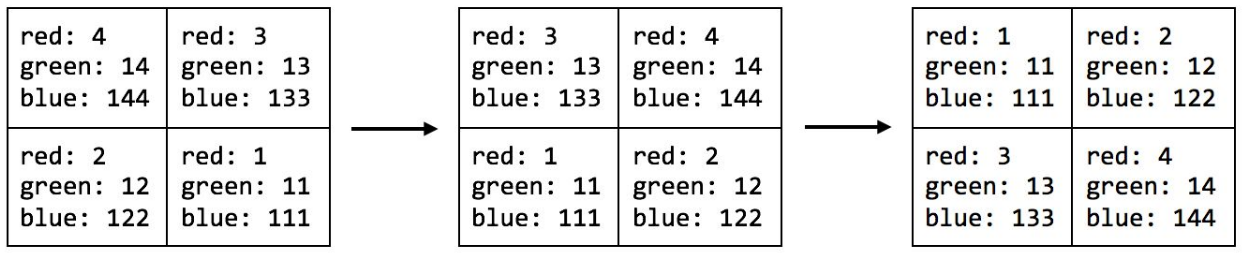
(example on next page)

For example, if you call:

```
>> [colorStructGrad] = colorGrad(colorStruct, 'red', 'blue');
```

The following 2x2 structure array should change as follows,

`colorStruct =` `colorStructGrad =`



red: 4 green: 14 blue: 144	red: 3 green: 13 blue: 133
red: 2 green: 12 blue: 122	red: 1 green: 11 blue: 111

red: 3 green: 13 blue: 133	red: 4 green: 14 blue: 144
red: 1 green: 11 blue: 111	red: 2 green: 12 blue: 122

red: 1 green: 11 blue: 111	red: 2 green: 12 blue: 122
red: 3 green: 13 blue: 133	red: 4 green: 14 blue: 144

(Sort rows based on red intensities) (Sort columns based on blue intensities)

Notes:

- The 'intensity values' will always be positive integers of type double on the range [0 - 255].
- The structure array can be any size, but will never be empty.
- The function `structShow()` has been given to show you what a given structure array looks like as an image. You may use it as reference to see if you're on the right track; however, it is not required to solve the problem.

Hints:

- `reshape()` will be helpful.

Function Name: `foodChain`

File Inputs:

1. *(char)* An Excel file of predators and prey

Outputs:

1. *(struct)* A 1x1 nested structure representing the food chain

Function Description:

Given an Excel file of predators in the first column and prey in the following columns, output a nested structure array where each structure has two fields: `Predator` and `Prey`, containing the full food chain represented by the Excel sheet. For example, if the Excel file input looked like this:

Chicken	Worm	
Human	Chicken	Goat
Worm	Bacteria	

The predators are in the first column (Chicken, Human, Worm) and the prey corresponding to each predator are in the subsequent columns. Your output in this case should be a nested 1x1 structure where the first layer's `Predator` field has `Human` in it and the `Prey` field has a 1x2 structure with the predators `Chicken` and `Goat`. The `Chicken` structure would have its `Prey` field populated by a 1x1 structure containing the predator `Worm`. The `Worm` structure would have its `Prey` field populated by a 1x1 structure with the predator `Bacteria`. The `Bacteria` and `Goat` structures would have their `Prey` field empty.

This might be difficult to visualize so you are highly encouraged to use the solution file on the test cases to see exactly what it expected of you.

Notes:

- There will only be one main predator so the output struct will always be 1x1.

- There will be no more than 4 layers to this food chain. (ex. Humans → Chicken → Worms → Bacteria)
- Predators and prey will not appear in any particular order
- If any of the predators have no prey, the `Prey` field should still exist but contain an empty string.