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 is 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. 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.

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.
- You can also use == to see which (if any) characters are incorrect.
- 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: `structSort`

Inputs:

1. *(struct)* A 1xN structure array

Outputs:

1. *(struct)* A 1xN structure array sorted by the length of the contents inside the field

Function Description:

Write a function called `structSort()` that will sort a structure array based on the length of the contents in a specific field. The field you are sorting by is the fieldname with the longest length. Once you have determined that field, sort the array in **descending** order based on the length of contents in that field. The contents are guaranteed to be all 1xM, but they may be of any data type you've learned so far (double, char, logical, cell, or structure). However, within a given field name, the data type will be constant.

For example, given a structure array where:

```
sa(1) =>                 sa(2) =>                 sa(3) =>
Name: 'Student A'        Name: 'Sdt B'            Name: 'StdentC'
Major: 'ChemE'           Major: 'CS'              Major: 'ECE'
GPA: '3.7'               GPA: '3.89'              GPA: '4.00000'
```

The array should be sorted by the length of the contents in `'Major'`, since that's the longest field name. The order to be sorted should be `[1 3 2]`.

Function Name: `movieStar`

Inputs:
1. *(struct)* A 1xN structure array of movie data

Outputs:
1. *(char)* The highest paid actor/actress
2. *(double)* The total amount of money he/she made
3. *(double)* The average movie rating of movies he/she appeared in

Function Description:

As a self-described movie buff, you enjoy spending your time doing all kinds of research about the film industry. This week, you've decided to do some research about how much actors and actresses are paid, using data from a structure array that you found on the internet.

The structure array will always have the following fields, and may have other fields.

Field name	Data within field
`Bad_Vegetable_Score`	A decimal percent between 0 and 1
`Budget`	A double
`Revenue`	A double
`Cast`	An Mx1 cell array of cast member names

An example structure can be seen below:

```
sa(1) =>
      Movie_Name: 'Puzzles'
      Year: 1984
      Genre: 'Mystery'
      Run_Time: '1h23m'
      Director: 'Kim Coppola'
      Budget: 5000000
      Revenue: 5500000
      Profit: 500000
      Cast: {'John Cena';
             'Stew Leonard';
             'Sam Walton';
             'Aunt Jemima';
             'Betty Crocker'}
      MPAA_Rating: 'PG-13'
      Bad_Vegetable_Score: .32
```

To calculate the highest paid actor, you should assume that the profit from the movie is split evenly between all of the actors/actresses in the movie. The profit of the movie is the difference between the budget and the revenue. If the movie lost money (revenue less than budget), then assume none of the cast members made any money. Once you have determined the highest paid actor/actress, you need to determine the average rating of the movies he/she was in. However, because you are assuming the highest paid actor/actress is a self-respecting individual, you should remove all movies in which Nicolas Cage is a co-actor before calculating the average movie rating. Additionally, if Nicolas Cage is the highest paid actor, you should assume there is a mistake in the data and output values for the second highest paid actor/actress.

Notes:

- The actor/actress money should be calculated before removing Nicholas Cage.
- An actor or actress' name will always appear exactly the same way in any movie in which they acted. Nicolas Cage will always appear as `'Nicolas Cage'`.
- An actor or actress will appear in the cast section only once per movie.
- In the case of a tie for highest paid actor/actress, select the actor/actress whose name appears first alphabetically.
- You should round every money calculation to two decimal places.
- Round the average rating output to two decimal places.

Function Name: `whereIsMySuperSuit`

Inputs:

1. *(cell)* A 1xN cell array, each cell containing a structure of information on a super
2. *(cell)* A 1xM cell array containing names of supers to update to `'Terminated'`

Outputs:

1. *(cell)* A 1xN cell array, each cell containing the updated structure of each super

Function Description:

You are Mirage, Syndrome's assistant. While he gets to design and build new technologies to destroy the supers he hates (superheroes), you have the more menial task of keeping track of the supers Syndrome has destroyed. The database of supers is tracked using structure arrays, and luckily for you, you are a whiz at MATLAB. You'll get this boring paperwork out of the way in no time!

Write a function that will take a 1xN cell array containing one structure in each cell and a 1xM cell array containing names of supers as inputs. Fields in each structure will describe a super. Each structure in the cell array represents the 'profile' of a super in the database. The structure will always contain the fields `'Name'` and `'Status'`, and may contain any number of additional fields. The cell array will always contain the names of the supers as they are formatted in the structure array. You must find the profiles of the supers specified by the cell array, and update their profiles. In order to update a profile, you must remove all fields other than `'Name'` and `'Status'`, and change the `'Status'` of the selected super to `'Terminated'` for each super specified in the cell array.

Notes:

- Each cell in the cell array will represent one super's profile.
- If a super is not terminated, the his/her structure should be unchanged.
- The order of the supers must stay the same (i.e. if the first input cell is Mr. Incredible's profile, the first output cell must be Mr. Incredible's profile).
- There may be supers in the 2nd input that are not in the 1st input - these may be ignored.

Function Name: `careerFair`

Inputs:

1. *(char)* A string for the filenames of resumes
2. *(double)* The number of resumes you must process

Outputs:

1. *(struct)* An 1xM structure array representing all the applicants
2. *(struct)* An 1xN structure array representing the selected applicants

Function Description:

As a new hire in your company's HR department, your first task is to attend a campus recruiting career fair. Overwhelmed by the stacks of resumes, you decide to write a function named `careerFair()` that will help you process through the large pool of applicants to select the best candidates.

You will be given a list of resumes as text files. The filenames will always be formatted as `<input1>##.txt`, where `<input1>` is the first input and `##` is the resume number. The second input specifies the number resumes you need to process through. For example, given `'Resume'` and `18` as the first two inputs, the files you need to work with would be `Resume01.txt`, `Resume02.txt` up to `Resume18.txt`.

In order to organize the applicants, you need to create a structural array where each structure contains information about one specific applicant. The fields of the structures are `Name(char)`, `GPA(double)`, `Education(char)`, and `Skills(cell)`. The information can be extracted from the resume file. If any of the fields exist in the text, it will be organized as `<Field>: Information`, where `<Field>` is one of the above categories, and `Information` is the data you must store inside that field. These headers may exist in any line in the text and in any order, but they are guaranteed to be present. The information should be stored as the class type specified - the contents of `Name` and `Education` should have no leading or trailing whitespaces. `Skills` will contain a comma separated list; each skill should be stored in a cell and also have no leading or trailing whitespaces. Because you are specifically interested in MATLAB experts, you will include an additional field `MATLAB(double)`, which will contain the number of occurrences of `'MATLAB'` in the resume. Your count should be case sensitive, and can include cases where MATLAB is a substring of another word.

For example, given a text file containing the following lines:

```
This is Waldo. I dare you to find me. #MATLAB
Name: Waldo
Address: Not found
Skills: Hiding, MATLAB
GPA: 4.0
Education: National Institute of Hiding
```

The structure for this applicant would be

```
applicant =
      Name: 'Waldo'
      Skills: {'Hiding'  'MATLAB'}
      GPA: 4
      Education: 'National Institute of Hiding'
      MATLAB: 2
```

Each applicant should be added to a structure array, which is the first output of the function. It will have an 1xM dimension, where M is the same as the second input (number of resumes to process).

Now that you have compiled your pool of applicants, it is time to develop the criteria to select the best candidates. You decide to use the candidate `GPA` as the baseline for comparison, given a few caveats. As a proud Yellow Jacket, if any applicant received his or her `Education` from `'Georgia Institute of Technology'` or `'Georgia Tech'`, multiply the `GPA` by 1.5 since you know how hard it is to earn a good GPA here. However, if any applicant received his or her `Education` from `'UGA'` or `'University of Georgia'`, multiply the `GPA` by 0.75. Lastly, add the number of occurrences of `'MATLAB'` found to each candidate's GPA. This will be a candidate's score. To select the best applicants, remove those who have a score less than the median of all the scores. Sort the remaining candidates by their score in descending order. This will be your second output.

Notes:

- You can use `'%02.0f'` instead of `'%d'` in `sprintf()` to format numbers as 2 digits.
- You will not be given more than 99 resumes.
- You can use `strtrim()` to remove leading and trailing whitespace from a string.
- Once you have the first output, you should not need iteration to generate the second output.