**Function Name:** `updateRoster`

**Inputs:**

1. (*struct*) An old roster containing players and their statistics
2. (*cell*) A MxN cell array containing the new roster with updated stats and players
3. (*cell*) A 1xP list of names of old players to be removed from the roster

**Outputs:**

1. (*struct*) A structure containing the updated roster

**Function Description:**

As the newly hired assistant for the Georgia Tech Athletic Association, you have been tasked with the laborious job of going through old rosters and updating them. With the power of MATLAB, however, you can make your work go smoothly!

You will be given a single structure (note: not an array) containing an old roster, a cell array that contains the data for the new roster, and a cell array containing a list of old players that must be removed. Each player will be represented in the structure as a field, with their name as a field. The content in each field is a cell array of statistics. The cell array of the new roster will have a header row as the top row of the array, and the names of the players are guaranteed to be in the first column.

Go through the old roster and update the players, add new players, and delete the players whose names are in the third input. In order to update a student's data, set the field equal to the entire row of statistics corresponding to that student in new roster, minus their name. Any player that is in the old roster, but neither removed nor in the new roster, should have `'Hall of Fame'` concatenated to the end of their statistics. Add fields for students in the new roster not currently in the old roster and assign each person's statistics to their new field.

For example, given the following old roster, updated array, and list to remove,

```
OldRoster =
      AasenGrant: {82, 'P', 6020}
    AdamsBrandon: {90, 'DT', 6020}
 AlexanderVictor: {9, 'LB', 5100}

UpdateArray =
      {'Name'      , 'Number', 'Position', 'Height';
       'AasenGrant', [    82], 'P'       , [  6020];
       'BryanWill' , [    70], 'OL'      , [  6040];
       'CampJalen' , [    80], 'WR'      , [  6020]}

Removed = {'AdamsBrandon'}
```

The output structure of the given data would be:

```
NewRoster =
      AasenGrant: {82, 'P', 6020}
 AlexanderVictor: {9, 'LB', 5100, 'Hall of Fame'}
       BryanWill: {70, 'OL', 6040}
       CampJalen: {80, 'WR', 6020}
```

**Hints:**

- Try to work from the old structure rather than trying to generate a new structure entirely

**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:** `getRose`

**Inputs:**

1. (*struct*) A 1xN structure array of contestants
2. (*cell*) A 1x(N-1) cell array containing different events

**Outputs:**

1. (*char*) The name of the winning contestant

**Function Description:**

Congratulations, you have just been chosen as the new Bachelor on the popular reality television show, *The Bachelor!* During each episode, you participate in activities with the contestants to see who you are compatible with. However, when it's time to choose who to eliminate at the end of each episode, you trust a computer program more than your own judgement. Write a MATLAB function that takes in a structure array of all the contestants and a cell array containing each episode's activity, and output the name of the winner.

The structure array will have the following fields: `Name`, `FirstImpression`, `Compatibility`, `FashionSense`, `MATLABSkills`, and `SobStory`. `Name` will contain each contestant's name formatted as a string, and all other fields will contain a double representing each contestant's score in that area.

During each episode, the person with the lowest score corresponding to that episode's activity will be eliminated. After the final episode, there should be only one contestant left, who will also be the winner. The activities and the scores important to that activity are as follows:

| Activity | Associated Score |
|---|---|
| `'Mocktail Party'` | `FirstImpression` |
| `'Group Date'` | `Compatibility` |
| `'Costume Party'` | `FashionSense` |
| `'Hackathon'` | `MATLABSkills` |
| `'Hometown Visit'` | `SobStory` |

Some scores will change throughout the season, as follows:

- If a contestant has the highest associated score during a mocktail party or a costume party, he/she will permanently gain a point in compatibility at the beginning of that episode.
- If any contestant's compatibility score is less than the average compatibility score during any episode, the contestant's score associated with **that episode's activity** will drop by one. After the episode ends, the score should return to its previous value.
- Whoever has the highest MATLAB skills score will gain a point in compatibility at the beginning of **every episode.**

**Notes:**

- If two contestants are tied for the highest score when determining score bonuses, they should both receive the score bonus.
- The point increase for `MATLABSkills` is cumulative. For example, after the third round, that contestant's `Compatibility` score would be three higher than what is was in the input.
- If multiple contestants are tied for the lowest score when determining who to eliminate, eliminate the one with the lowest sum of **all** scores.
- Only one contestant will be eliminated each episode: no two contestants with the same minimum score for an event will have the same total score.
- Some activities will be repeated.

Do not round the average compatibility score.

**Function Name:** `runHomeworkCode`

**Inputs:**

1. (*struct*) A 1x1 structure containing information about a MATLAB problem

**Outputs:**

1. (*struct*) The same structure from the input, but with some slight changes

**Function Description:**

Congratulations, you just got a job as a MATLAB TA! You are writing a homework problem for your students, and you have stored all of the information for a student's function in a structure containing three fields. The first field is `lines`, which contains all the lines of code for the given function in a 1xN cell array (where each cell is different line from the function). The second field is `name`, which simply denotes the name of the given function. The final field is `inputs`, which will contain a 1xM cell array that contains the different test case inputs of the function, where each cell contains a single test case input.

Your job is to create a new field called `outputs`, which will be 1xM cell array containing the results of running the given function on each of the inputs. But, since the function file is not already given, you will have to use the `lines` field to create the function file so that you can evaluate it for each of the given `inputs`! The order of `outputs` in the cell array should correspond to the order of `inputs`. Here is what to do:

1) Create a function `.m` file in the current directory using Low-Level File I/O.
    - Write a function header. The header will always be written to have one input called "`in`", and one output from the function called "`out`".
    - Write each string from the cell array stored in the `lines` field into the new `.m` file. Make sure to suppress the outputs of each line, and don't forget the new line characters!
    - Don't forget to put an '`end`' at the end of the file! Close the file handle once you're done writing.
2) Create a new field in the structure called `outputs` that contains the corresponding results of running the newly created function on each element stored in `inputs`.
    - In order to run your function once it is written, you will need to use a very cool MATLAB function called `feval()`.
    - `result = feval('<nameOfFunction>', <inputToFunction>)` simply calculates the output of running the function with the name '`<nameOfFunction>`' (as a string) on the input `<inputToFunction>`.
    - For example, `feval('sum', [2 3 4])` outputs the value `9`. You can read more on this function here.
3) Delete the `lines` field from the structure since it is no longer needed.

After doing these steps, output the new structure. It should now have the field `outputs`, and be missing the field `lines`.

Example:

```
structIn =
      name: 'add2AndSquare'
    inputs: {[2]  [1]  [7]  [5]  [3]}
     lines: {'x = in + 2',  'out = x.^2'}
```

should become...

```
structOut =
      name: 'add2AndSquare'
    inputs: {[2],  [1],  [7],  [5],  [3]}
   outputs: {[16],  [9],  [81],  [49],  [25]}
```

**Notes:**

- The input to the function will always be called `'in'`, when referenced in the `lines`. The output of the function will always be called `'out'`. Therefore, the function header will always be the same.
- The input and output of the the function could be any size or type.
- When writing the .m file, you should put semicolons after every line to suppress the code.
- Naturally, a new file will be generated from your code, since you have to run it. However, this file will not be graded, only the output structure, so you don't have to delete the new file.

**Hints:**

- While you have never done this before, writing to a `.m` file is no different from writing to a `.txt` file. The only difference is that when creating the new file, your filename should be `'filename.m'` instead of `'filename.txt'`.
- `feval()` will take in a single input and give you a single output. You cannot try to evaluate the entire cell array with multiple inputs at once.
- Do not include the `'.m'` as part of the first input to `feval()`.

## Extra Credit

**Function Name:** `helpDesk`

**Inputs:**

1. (*struct*) A 1xN structure array containing each TA's help desk hours
2. (*char*) The name of a text file containing each TA's problem preferences
3. (*char*) The name of an excel file containing the data on the students at help desk

**Outputs:**

1. (*struct*) A structure array containing the names of the students each TA will help

**Function Description: *Please read the entire Notes section carefully***

Thanks to the helpful feedback on end-of-semester CIOS surveys, a big change is coming to help desk for CS1738 at a nearby university. Upon arrival at help desk, students will fill out a short form and then be assigned to the TA who can best help them. The form will collect data on the time the student arrived at help desk, their name, the problem number they are stuck on, and a brief description of their problem, in that order. This data will be compiled into an excel file, the third input, where each column contains one of those four pieces of information. The first row of the excel file will always contain the corresponding headers: `'Time'`, `'Name'`, `'Problem'`, `'Description'`. The filename will always be formatted as `'<Day of week><Date><Month>Helpdesk.xlsx'`. The day will match the formatting of the structure fields, the date will be just the two digit day number (01-31), and the month will be the common abbreviation (Jan, Feb, etc.). The arrival time will always be formatted, for example, as `'2:30PM'`. The problem number will always be an integer of type double between one and five, inclusive.

The input structure array will have the six fields: `Name`, `Monday`, `Tuesday`, `Wednesday`, `Thursday`, and `Friday` in that order. Each field, apart from the first which will contain the TA's name, will contain a string describing the hours the TA works at help desk. An empty string means the TA has no hours that day. If a TA has multiple non-consecutive hours on a single day, the range will be separated by a comma. A sample structure from the input array:

```
      Name: 'Paul Johnson'
    Monday: ''
   Tuesday: '3:00-4:00'
 Wednesday: ''
  Thursday: ''
    Friday: '2:00-3:00,4:00-5:00'
```

The text file will contain a TA's name followed by the order in which they are best at helping with the homework problems, with their top preference first. There will always be five homework problems each week. For example, a line in the file could be:

```
'Paul Johnson:3,2,5,4,1'
```

In order to truly optimize the help desk experience, students should be assigned to the TA who is best at answering questions about the problem that they need help with, if possible. However, TA's have lives too! So time is a major factor. A TA will take 5 minutes to help a student on the problem the TA is best at helping with, 10 minutes for the second best, 15 for the third, then 20, and finally 25 minutes for the problem they are worst at helping with. This order is based on the information from the TA problem preferences text file.

To assign students, you should first look at the total time that has already been allotted to each TA up to that point. For example, we will consider the following three TAs for the rest of this description:

```
'TA1:1,2,3,4,5'
'TA2:4,5,3,2,1'
'TA3:2,4,5,1,3'
```

If TA1 is already helping two students with problem 1, TA2 is helping two students with problem 4, and TA3 is helping one student with problem 5, then their respective time allotments at the moment are 10, 10, and 15 minutes. You should then assign the next student to the TA whose total time allotment would be the lowest with that student. So if the next student needs help with problem 5, you should assign the student to TA2 since their time allotment would become 20 minutes, whereas TA1 would have been 35 and TA3 would have been 30. Notice that even though TA3 is only helping one student, we assigned this new student to TA2 since that was the most efficient timewise. In the case of a tie for which TA's time allotment would be lowest, assign the student to the TA that appeared first in the input structure array, not the TA with the fewest students.

It is important to note that no TA will stay at help desk longer than they are supposed to. If all three TA's were supposed to work 2-3 and each currently had time allotments of 55 minutes, then if the next student needed help with problem 3, he or she would need to be assigned to a TA that would be working at help desk from 3-4, even if the student had arrived before 3. Adding the student to any of the current TA's would bring their allotment to over 60 minutes, which is not allowed. Also, if a student were to arrive at help desk at 5, and all the TA's that were there from 4-5 only had an allotment of 20 minutes each, the student would still need to be assigned to a TA that works 5-6 since the 4-5 TA's will leave.

The final output should be a structure array containing the six fields: `Name, Problem1, Problem2, Problem3, Problem4, Problem5.` Each structure should contain the TA's name in the first field, and a cell array column vector of each student that will be helped on each problem for the other fields. Every TA that helps at least one student at help desk should be included in the structure array. A possible output structure is:

```
     Name: 'Paul Johnson'
  Problem1: []
  Problem2: {'Justin Thomas'}
  Problem3: []
  Problem4: []
  Problem5: {4x1 cell}
```

**Notes:**

- Every TA is scheduled for exactly three hours of help desk each week.
- Help desk is open 2-8 MTWTh, and 2-5 F.
- There will always be at least one TA at help desk during an open hour.
- If a TA is at help desk for multiple hours in one day, their allotment resets each hour. For each hour, the maximum allotment is 60 minutes, inclusive.
- Once the first student has been assigned to TAs for the next hour, no more students should be assigned to the TAs from the previous hour.
- TA's help desk hours will always start and end on the hour. The start time is inclusive, the end time is exclusive.
- Every TA in the structure array will appear in the text file, but not every TA in the text file is guaranteed to be in the structure array.
- If a TA is at help desk during the given hours, but does not help any students, they should not appear in the output structure array. A given TA should only appear once in the output structure array, even if they work multiple hours.
- The name field in the output array should contain type char. Student names in the output array should all be in a cell array, even if there is only one. If there are no students, there should be an empty array, not an empty cell.
- As long as the student arrives at help desk before a given TA who works for one hour leaves, and the TA's time allotment would not exceed 60 minutes by helping that student, the TA will help even if they have to technically stay past their end time. So if a student arrived at 4:55 and needed help with the hardest problem, a TA who was supposed to leave at 5 would help as long as their time allotment at the moment was 35 minutes or less, even though they are technically staying at help desk until 5:20.
- The students can spill over from one hour of help desk into another, but will never spill into another day. It is guaranteed that all the students in the excel file can be helped before help desk closes for the day. As such, the excel file will never represent more than a single day of help desk.
- This method of assigning students is not guaranteed to actually minimize the total time needed to help the students. However, it is more fair about letting students who arrived first get help first, so be sure to follow this method.

**Hints:**

- Use helper functions!
- Cell arrays will be very useful for storing your data.