Drill Problem #1

Function Name: zoesStrizzatta

Inputs:

1. (*struct*) A 1xM structure array with fields `'Ingredients'` and `'Costs'`
2. *(cell*) An Nx3 cell array of vegetarian substitutions

Outputs:

1. (*struct*) The updated structure array with the ingredients

Function Description:

You are a chef at a local restaurant, and everybody always raves about your most famous dish, the Strizzatta. A Strizzatta is very similar to a Stromboli made from pizza dough, but it has a white sauce with various ingredients rolled within. To your dismay, you have never been able to try this dish because you are vegetarian. One day you decided to make a vegetarian version of the Strizzatta, and it was so delicious your boss told you to write down the ingredients so they could add it to the menu. Being the savvy CS 1371 student you are, your first thought was to use MATLAB to do so. You decide to take the meat items from the list of ingredients and replace both the ingredient and the price of the ingredient with the vegetarian option and its respective price.

Write a function, `zoesStrizzatta()`, that takes in a structure array with fields `'Ingredients'` and `'Costs'`, as well as an Nx3 cell array where the columns of the cell array correspond to a list of the current meat item, its respective vegetarian substitute, and the cost of this substitute, in that order. To create the new dish for the menu, you will need to replace all the meat ingredients in the structure array with its vegetarian option and the cost for that vegetarian option.

Notes:

- The structure of the ingredients and the cell array of substitutes are not guaranteed to be the same length.
- The fields `'Ingredients'` and `'Costs'` are not guaranteed to be in any order.
- Any meat items in the structure array are guaranteed to have a vegetarian substitute, and there may be multiple meat items called for in the recipe.

Drill Problem #2

Function Name: myStruct

Inputs:

1. (*cell*) A 1xN cell array containing fieldnames and data

Outputs:

1. (*structure*) The contents of the input cell array in the form of a structure

Function Description:

Write a MATLAB function that creates a structure from a given cell array. The cell array will be formatted in the following way:

```
inCellArr = { <field 1 name>, {cell array of field 1 contents}, <field 2 name>, {cell array of field 2 contents}, <field 3 name>, {cell array of field 3 contents}, <field 4 name> ...}
```

Take the cell array and create a structure array with fields given by the strings in the oddly-numbered indices and with field contents given by the field content cell arrays in the evenly-numbered indices. Each element in the field content cells should be stored individually in the corresponding index of the structure (i.e. if the field content cell arrays are 1xM, then the output structure should be 1xM). The field content cell arrays will all be the same size, with the singular exception of a 1x1 cell. In this case, that single element should be assigned to that field name for every index of the structure (i.e. if your structure is 1xM, then all M elements for this field will be the same piece of data, which is the contents of the singular cell).

Notes:

- **You may NOT use the `struct()` function**. The premise of this problem is for you to thoroughly learn how it works by coding it yourself. Any submission that uses the `struct()` function will receive 0 credit.
- The input cell array will always have an even length.
- The contents cell array for the first field is guaranteed to be a 1xM cell and defines the size of the output structure.
- You must always assign the contents of a 1x1 cell array into each index of the output structure, not the 1x1 cell.

Drill Problem #3

Function Name: structDisp

Inputs:

1. (*struct*) An MxN structure array

Outputs:

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

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 displayed like an array of MATLAB’s display for a 1x1 structure. You have been given a helper function `struct2charArr()`, which takes in a 1x1 structure and turns it into an RxC character array, where R 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 C is the minimum number of columns needed to fully capture the data in the fields. On the other hand, if all of the fields in the structure contain short data entries, C may be small. The easiest way to see what the helper function outputs is to try it with different 1x1 structures. With this function in your Current Directory, you can call `struct2charArr()` in your code as if it were a build-in function. So if there is a long vector contained in one of the fields, C may be very large.

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 `brickLayer()` 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.

Drill Problem #4

Function Name: mostWanted

Inputs:

1. (*struct*) A 1xN structure array
2. (*double*) The number of miles Agent Z is allowed to travel

Outputs:

1. (*struct*) A 1xM structure array listing the criminals captured
2. (*double*) The number of miles traveled

Function Description:

You have completed your degree at Georgia Tech, and Interpol has hired you as their next Crimes Analysis Technician, or CAT. Your first task is to create a travel itinerary for a secret agent—Agent Z—to assist in this agent's criminal hunt, while also maximizing the agent's travel efficiency (Interpol promotes saving Earth's natural resources as well as catching criminals). Criminals are given a score by Interpol saying how important their capture is; this score ranges from 10 to 100, where 10 is the least wanted and 100 is the most wanted. For each criminal, you know a name, an approximate location in Country X, and how "wanted" the criminal is. Write a MATLAB function, `mostWanted()`, to maximize Agent Z's accumulated "score" from capturing criminals, while keeping your agent under the agent's allotted travel mileage. The function should input a structure array of criminals with the field names `'Name'`, `'Location'`, and `'Points'`, and output a structure array (with the same field names) of the criminals captured, as well as output the total distance travelled along Agent Z's itinerary route.

Agent Z will start at a "neutral zone" with 0 miles traveled. Country X has 5 regions, and no matter what region someone is currently in, it always takes the same distance to travel from one region to a given region. The travel distances to those specific regions are as follows:

Location (As found in the structure)	`'NE'` (North East)	`'SE'` (South East)	`'MW'` (Midwest)	`'NW'` (North West)	`'W'` (West)
# Miles	200	300	500	600	700

For example, if someone were in the North West region (`'NW'`) and wanted to travel to the North East region (`'NE'`), it would take 200 miles. Further, no matter what region someone is in, it would take 700 miles to travel to the West (`'W'`) region.

To decide the next criminal to capture, check the two next highest wanted criminals in the given structure array (or, at the very start, the two highest wanted criminals in the structure). Between those two criminals, the criminal with the smaller travel distance will be the next criminal on the list to capture, regardless of how "wanted" they are (choose the more wanted of the two in the case of a tie in the travel distance to either criminal). A criminal located in the same region of Country X as Agent Z's current position will only take 50 miles of travel to reach.

Drill Problem #4

The agent will continue to traverse Country X in search of the next most wanted criminal for both as long as Agent Z has miles left to traverse and as long as one of the top two wanted criminals are within that travel restriction.

The output structure array will have the same fields as the input structure array, and the values of in those fields for a captured criminal should be identical to what is found in the input structure array. The output structure array should be in the order of criminals captured; the first criminal captured should be the first structure in the structure array, and the last criminal captured should be the last structure in the structure array.

Notes:

- You are guaranteed to be able to capture at least one criminal.
- This algorithm is not by any means a perfect way to maximize Agent Z's travel efficiency.

Hints:

- You may find a helper function useful.
- Think about how manipulating the input structure can help simplify finding the two highest wanted criminals at any point.
- Consider building the output structure array as the criminals are being captured.

Drill Problem #5

Function Name: thriftyHalloweener

Inputs:

1. (*struct*) An MxN structure array containing information about stores, costume items, and cost.
2. (*char*) A string containing the name of the character you want to be for Halloween.

Outputs:

1. (*cell*) An Lx3 cell array containing the costume items, store, and price.
2. (*double*) The total cost of the costume.

Function Description:

As Halloween is approaching, you suddenly have to perform the annual scramble to find parts for a killer costume. Being in college, you want to try to find the most economical way to put together your costume, and you are going to use your programming skills to help you out!

Write a function in MATLAB called `thriftyHalloweener` that will take a structure array containing the costs of your costume items and where to get them and a string containing the name of the character you will want to be. The items you need for your costume will be contained in an Excel file whose name will be the character's name in camel case, i.e. if you are going as 'Draco Malfoy', the items you need for the costume are in 'dracoMalfory.xls'. The items will be listed in the first column of the Excel file, with no column header. The structure array contains two fields: 'Name' and 'Inventory'. The 'Name' field contains the names of the stores you will be shopping at, and the 'Inventory' field contains another structure array of the items and prices offered at the corresponding store, under the fields 'Item' and 'Price', respectively. You will need to identify which store offers the items you need at the cheapest price.

You will output a cell array containing 3 columns. The first column is the costume items, the second column is the store offering the cheapest price, and the third column is this price. Each column should have the header 'Item', 'Store', and 'Price', respectively. Additionally, you will output the total cost of the costume, assuming you get the cheapest price on all items.

Notes:

- The order and case of the items in your output cell array should match the order and case of the items in the 'Inventory' structure.
- All items will be found in at least one store, but each store does not sell every item.
- There will be no ties for the store offering the cheapest price.

Hints:

- You can use the `structDisp()` function from problem #3 to visualize the structures.
- You can use a modified version of the `camelCase()` function from Homework 5 as a helper function.

Drill Problem: EXTRA CREDIT

- THIS PROBLEM IS EXTRA CREDIT! -

Function Name: magicTheStructuring

Inputs:

1. (*struct*) A 1xM structure array containing the names of the players in the game as well as the cards they have on the field
2. (*char*) A string containing the name of the player who is attacking

Outputs:

1. (*struct*) A 1x1 structure of the graveyards for both players
2. (*double*) The amount of damage taken by the defending player

Function Description:

You and your friends have found that you have too much work to do at Georgia Tech, but you still want to play your favorite card game: Magic: The Gathering. You've decided to create a function that will take an input structure of each player's name and the creatures that they have on the field and an input string of the name of the attacking player; this function will determine which creatures will die when the defending player blocks, as well as what damage (if any) will be taken by the defending player.

As a general preface, Magic: The Gathering is a card game with creatures, sorcery spells, artifacts, manna, and many, many other special situations that make it a very dynamic and strategic game. For the purposes of this function, you will solely deal with the creature dynamic of the game. Each player starts off a game with a specific, predetermined amount of life points. Throughout the game, each player will build up how many creatures they control, and they can attack each other in hopes of dealing damage to the other player's life points. When one player eliminates the other player's life points, that first player wins. This function will simulate one attack-block-damage exchange (or attack phase) between two players.

A quick aside for all you actual Magic players: this problem takes into account no keyword abilities (assume all creatures are vanilla), and there is no double blocking. This is a very simple "attack, block, damage" simulator. Permit yourself a small suspension of disbelief.

There are essentially three ways an attack phase can occur: the attacking player has one creature or multiple creatures, and the blocking player has no creatures; the attacking player has one creature, and the blocking player has one creature; or the attacking and/or blocking players have multiple creatures.

Scenario 1: Blocking Player has No Creatures

If the blocking player has no creatures, then the attacking creatures will be able to directly deal damage to the blocking player, and the damage dealt with be the sum total of the attacking creatures' "power."

Scenario 2: Attacking and Blocking Players have One Creature

When the Blocking Player has one (or multiple) creature(s), they can choose which creature will block which attacking creature (only one creature can block each attacking

creature). When both players have one creature and the blocking player chooses to block, there will be an interchange of damage in this attack phase. Each creature has “power” (for dealing damage) and “toughness” (for defending damage). Once a creature is blocking, the attacking creature will deal its power to the blocking creature’s toughness, and the blocking creature will deal its power to the attacking creature’s toughness. This damage is “stacked” simultaneously. If a creature is dealt damage (the opposing creature’s power) more than or equal to its toughness, that creature is killed and sent to the “graveyard.” It is possible for both creatures to be killed in this damage exchange.

Scenario 3: Attacking and Blocking Players have Multiple Creature(s)

In the case of multiple creatures, the blocking player will strategically choose which blocking creatures will block which defending creatures. Once blocking creatures have been assigned, damage between each of the creatures will be dealt in the same way as in Scenario 2. In the event where there are more attacking creatures than blocking creatures, then any creatures who are not blocked will deal damage directly to the blocking player. In the event where there are more blocking creatures than attacking creatures, only enough creatures to block will block, and the rest will remain unused.

Given this summary of an attack phase, it is time write a function for this! The input structure will contain five fields: `'Type'`, `'Name'`, `'Owner'`, `'Power'`, and `'Toughness'`. The table below describes the options for each field:

<table>
<tr><th>Fieldname:</th><th>'Type'</th><th>'Name'</th><th>'Owner'</th><th>'Power'</th><th>'Toughness'</th></tr>
<tr><td rowspan="3">Contents:</td><td>'Player 1'</td><td>(char) Player 1 name</td><td rowspan="2">'N/A' (Players)</td><td rowspan="2">'N/A' (Players)</td><td rowspan="2">'N/A' (Players)</td></tr>
<tr><td>'Player 2'</td><td>(char) Player 2 name</td></tr>
<tr><td>'Creature'</td><td>(char) Creature name</td><td>(char) Player 1 or 2 name</td><td>(double) creature power</td><td>(double) creature toughness</td></tr>
</table>

In any given phase, all of the attacking player’s creatures are going to attack, and the defending player’s creatures are going to defend. If the damage dealt to a creature is greater than or equal to its toughness, it is killed and the killed creature’s structure element from the input structure array should be placed in the output graveyard structure.

When assigning a blocker to an attacker, the creature with the lowest power should block the creature with the highest power. If there is a tie between the powers of two attacking creatures, the one with the lowest toughness should be blocked first, then the one with higher toughness next. The opposite is true for blockers: if a blocking creature has the same power as another blocking creature, the one with the higher toughness should block the higher power attacker.

Drill Problem: EXTRA CREDIT

For Example:

As an aside, power and toughness will be given in this example as their integer values separated by a slash. A 3 power, 4 toughness creature will be expressed as a 3/4 creature.

Player 1 has a 1/1, a 1/2, and a 3/3 creature.
Player 2 has a 5/6, a 5/5, and a 4/1 creature and is attacking.

	Pair 1	**Pair 2**	**Pair 3**
Attacking Creature	5/5	5/6	4/1 (DEAD)
Blocking Creature	1/2 (DEAD)	1/1 (DEAD)	3/3 (DEAD)

In this case, there are enough blockers for each attacker, and Player 1 will take no damage. If, however, there were only two creatures under the ownership of Player 1, Player 1 would take 4 damage, because the two higher power creatures would be blocked and the lowest (the 4/1) would deal damage to the player equal to its power. If there are more blocking creatures than there are attacking creatures, the lowest power (and potentially toughness) creatures should not block, only those with the highest power.

Output the dead creatures in a graveyard structure; if no creatures die, the graveyard structure output should be an empty vector. In the graveyard structure, the creature owner's name (Player 1 or Player 2's name) is the field name, and the value inside the field is a nested structure array of all the killed creatures' original structures. The graveyard structure field names should be in alphabetical order. The creatures in each graveyard field should be organized by power and toughness, with the lowest power first, and the highest toughness prioritized when powers are the same. Further, output the damage dealt to the defending player, if any was dealt; if no damage was dealt, output 0.

Notes:

- It is **HIGHLY** recommended that you take a look at and familiarize yourself with both the inputs and the outputs of the solution code. The exact formatting is relatively simple, but ensuring that your output graveyard matches the solution code graveyard is crucial.
- The function `orderfields()` may be helpful.

Hints:

- You will need to use dynamic field naming to create the names of the fields of the output structure array.