Function Name: `celery`

Inputs:

1. (*cell*) MxN cell array of nutrition facts for different foods

Outputs:

1. (*struct*) 1x(M-1) structure array containing those same nutrition facts

Banned Functions:

`cell2struct`

Topics: (*structures*), (*cell arrays), (iteration), (sorting)*

Background:

Lucky the Leprechaun loves his job. He gets to carry his pot of gold all the way to the end of the rainbow every year on a very special day. Lately however, he has noticed that he seems to be getting tired more easily and seems to struggle to get to the end. Perhaps it is all the chocolate coins and four-leaf clovers that he is eating. With the spring equinox on the horizon (which is basically New Year's for leprechauns), Lucky resolves to get back in shape, and luckily for him, one of his friends gifted him a data structure of nutritional information to celebrate the most important holiday of March, National Celery Month! Unfortunately, all his nutritional info is jumbled up in cell arrays (his friend did not pay attention when they taught structures in 1371), which are no better than runes to Lucky. You decide to help Lucky convert all his cell arrays to structures (in return for some of his gold, of course).

Function Description:

Given an MxN cell array, convert the cell array into a structure array. The first row of the cell array will contain the field names to use in the structure array. Each **row** after the first contains information for a single food item. There will always be a `'Rating'` column containing the health rating of the foods. The values in the `'Rating'` column will lie between 1 and 10 (inclusive, 10 being the healthiest). The structure array should be sorted by health rating, from most healthy to least healthy (numerically descending).

Example:

```
cellFoods = {'Carbs', 'Name', 'Rating', 'Calories';
             44, 'Four-Leaf Clovers', 4, 444;
              9, 'Chocolate Coins', 2, 80;
             19, 'Celery', 9, 70};
structFoods = celery(cellFoods);
```

`structFoods →`

Carbs:	19	44	9
Name:	'Celery'	'Four-Leaf Clovers'	'Chocolate Coins'
Rating:	9	4	2
Calories:	70	444	80

Notes:

- The location of the `'Rating'` column is not guaranteed.
- There may be ties for health rating. In these cases, let MATLAB's internal decision making decide the order of ties.
- There will always be at least one food item.

Hints:

- Recall that a structure's field names have no particular order.
- The `cell2mat()` function may be useful.

Function Name: `luckyCharms`

Inputs:

1. (*cell*) A 1xM cell vector with the customer's order details and values
2. (*struct*) 1xN structure vector representing your invoice book with information about all of your orders

Outputs:

1. (*char*) A string describing the order that the customer is here to pick up

Topics: (*structures*), (*indexing structures*), (*iteration*), (*cell arrays*)

Background:

You're the owner of a local flower shop in Ireland and business has been booming this week! You've been flooded with hundreds of orders for bouquets of clovers. It's a bit strange that people are so passionate about clovers, but you assume it has to do with the fact that the Spring Equinox and the International Day of Forests are right around the corner. Customers are constantly coming in to retrieve their bouquets, so you've decided to turn to MATLAB so you can assist each customer as quickly as possible!

Function Description:

You are given a 1xM cell vector (input 1) containing once-nested, 1x2 cells with the current customer's order. The first index of each nested cell will contain an item, and the second index will contain the value for that item, specific to the customer's order. Each item will correspond to a fieldname in the given 1xN structure (input 2), whose indices correspond to particular invoices.

Using the customer's order details and items, search the structure vector for the structure that matches the current customer's order. Once you find the correct structure, obtain the value in the field `'Order'`, and output a string with the following format:

```
'This customer is here to pick up Order Number <order number>!'
```

Example:

```
customerInfo = {{'Leaves',4}, {'Clovers', 120}};
invoices =
```

Clovers:	120	45	120
Order:	33	12	99
Leaves:	3	4	4
Price:	18	8	22

```
str = luckyCharms(customerInfo, invoices);
```

```
str →
      'This customer is here to pick up Order Number 99!'
```

Notes:

- All values in the structure will be type double.
- The customer's order details will only correspond to one invoice, and the customer's order is guaranteed to exist within the invoice structure
- The invoice structure's fields are not guaranteed to be consistent across test cases, with the exception of the `'Order'` field.
- All fields found within the customer's order details can be found within the invoice structure.

Hints:

- Think about how you can use masking to compare the customer's order details to those stored in the invoice structure array.
- Consider deleting non-matching structures as you search for the correct structure.

Function Name: `nutTheRightTaco`

Inputs:

1. (*struct*) 1x1 structure with N fields representing tacos and their ingredients

Outputs:

1. (*struct*) 1x1 updated structure with all the peanut butter tacos removed

Topics: (*iteration*), (*removing fields*), (*cell arrays)*

Background:

It's March, which means that it's time for ~~Saint Patr~~ National Crunchy Taco Day! You're stoked, so you start preparing your tacos for the celebrations. You've created *amazing* tacos, but a mischievous leprechaun ruined some of your tacos by adding peanut butter! Not peanut butter and flour tortillas again! You're relying on MATLAB to make things right for the best holiday in March.

Function Description:

Create a function that removes peanut butter tacos from a structure. Each field of the structure contains a cell array of ingredients for a different taco. The ingredients will be character vectors contained in individual cells. Go through the cell array to determine if the taco contains peanut butter. In the cell array, peanut butter is represented as `'peanut butter'`, (case insensitive). Remove any peanut butter-containing taco from the structure (cause yuck 🤢). Output the updated structure.

Example:

```
yucky =

  taco1: {'tortilla', 'lettuce', 'cheese'}
  taco2: {'beef', 'tortilla', 'peANut butTer', 'cheese', 'onions'}
  taco3: {'tortilla', 'cheese', 'tomatoes', 'onions', 'sour cream'}

yummy = nutTheRightTaco(yucky);

yummy →

  taco1: {'tortilla', 'lettuce', 'cheese'}
  taco3: {'tortilla', 'cheese', 'tomatoes', 'onions', 'sour cream'}
```

Notes:

- It is guaranteed that at least one peanut butter taco will be present in the structure.

- There will always be at least one taco without peanut butter as well.
- Peanut Butter in the ingredients cell array will be represented only as `'peanut butter'`. However, the letters of this string can be in any case, lower or upper.
- Each taco may have a different number and kind of ingredients. The cell arrays are not guaranteed to be the same length.
- There may be any number of fields in the given structure, and the fieldnames may not be consistent across test cases.

Function Name: `stTacoDay`

Inputs:

1. (*struct*) a 1xM structure vector representing the ingredients for each person

Outputs:

1. (*char*) string containing the vegetarian's name and taco toppings

Topics: (*nested structures*), (*conditionals*), (*iteration)*

Background:

You've been selected as the person to go to tech square and get everyone's Moe's order to celebrate Taco Day. One of your friends is a LOYAL vegetarian and refuses to even touch a container containing meat, while you and the rest of your friends are avid devourers of seared meat and will eagerly be including some in your orders. However, the Moe's worker doesn't know what a vegetarian is so you decide to write them a MATLAB function that allows them to figure out who is the vegetarian just from the orders!

Function Description:

Given a 1xM structure representing each person's order, go through each person's taco toppings and figure out who is the vegetarian. The vegetarian is the only person in the structure array who does not have `'meat'` as one of their toppings.

The outermost structure will contain 3 fields: `name`, `topping`, and `nextTopping`. The `nextTopping` field will contain either a 1x1 nested structure with the fields `topping` and `nextTopping`, or an empty vector (`[]`). Unnest the nested structure in the `nextTopping` field until you reach the end of the toppings list for that person. The end of the toppings list is indicated by an empty vector in the `nextTopping` field.

Once you have found the vegetarian, output their name and toppings in a formatted string according to the format below.

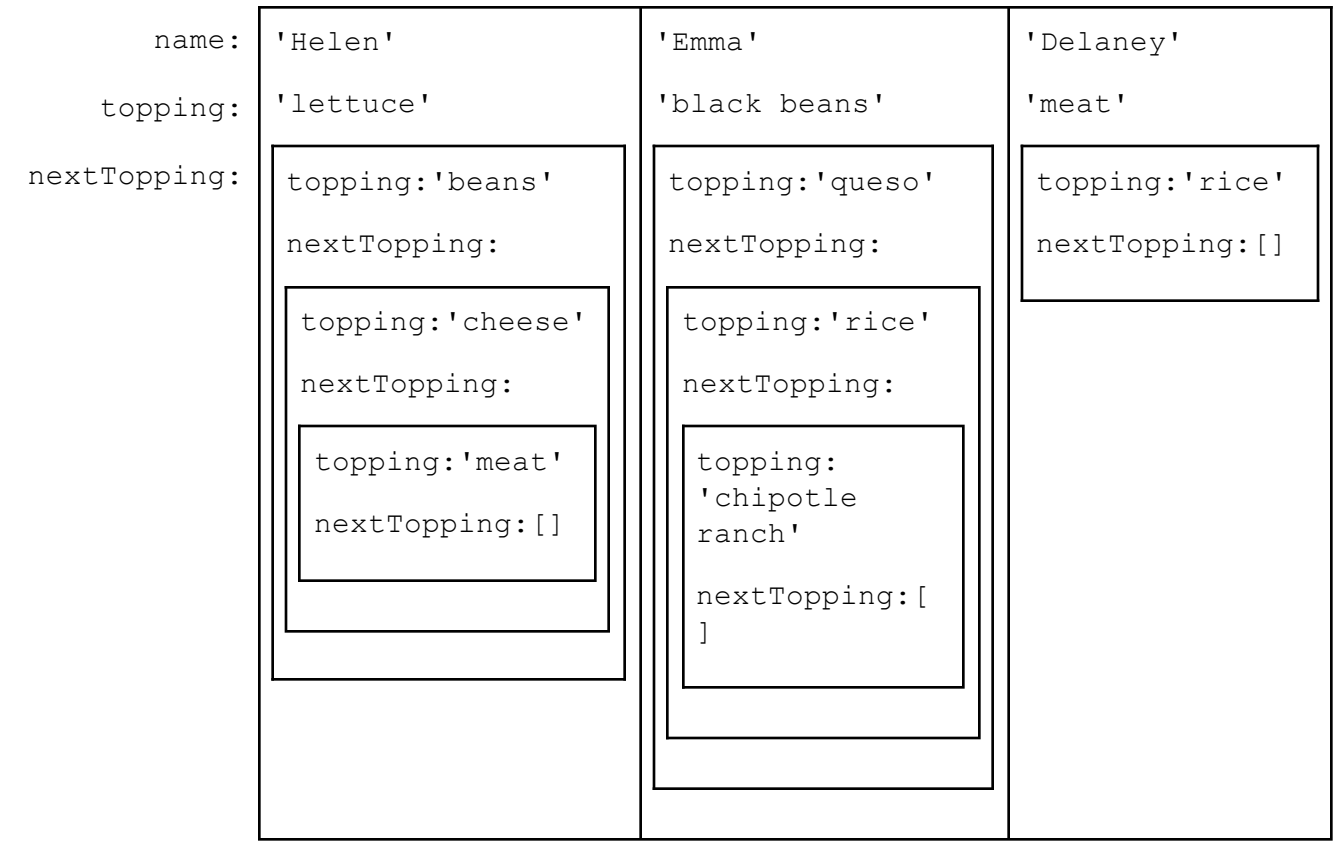
```
'<vegetarian's name> is vegetarian. They got <list of
                toppings> on their taco!'
```

Each topping should appear in the order they appear in the structure, and be separated by a comma and single space. For example, if a particular vegetarian had K number of toppings, the `<list of toppings>` would look like:

```
'<topping 1>, <topping 2>, ..., <topping K>, '
```

Example:

`orders =`




```
veggieStr = stTacoDay(orders);

veggieStr →
     'Emma is vegetarian. They got black beans, queso, rice,
     chipotle ranch, on their taco!'
```

Notes:

- There will always be **only one** person without `'meat'` as a topping.
- Meat will only appear as `'meat'`, so you don't need to search for words like `'beef'`.
- The word meat will not be contained within another word, such as `'meatless beef'`.
- Every person is guaranteed to have at least one topping.
- The `'name'` and `'topping'` fields are guaranteed to have values of type char.
- Even if the vegetarian has only one topping, include the comma and space like normal.

Hints:

- The `isempty()` function may be useful.

Function Name: `corndogWars`

Inputs:

1. (*struct*) 1x1 structure containing the original stats of fighter A
2. *(struct)* 1x1 structure containing the original stats of fighter B

Outputs:

1. (*char*) A string describing the outcome of the battle
2. *(struct)* 1x1 structure containing the updated stats of fighter A
3. *(struct)* 1x1 structure containing the updated stats of fighter B

Topics: (*iteration*), (*fieldnames*)

Background:

You've been looking forward to it all year, and it's finally time. Tired and starving, you get off work this third Friday of March, pack your bags, and you're off to celebrate this ancient March holiday in the land it came from, a land of fields of green and pots of gold. Of course, we mean the green fields of Corvallis, Oregon, where in the ancient forgotten year of 1992 two men set out in search of a pot of glistening golden cooking oil to celebrate the very first National Corndog Day.

The festivities in Corvallis were always a joy to behold, but this year things are different. People's love for corndogs seems to have grown so intense that it has escalated to all out war between the many factions of corndog connoisseurs. To put an end to this violence, you suggest an alternative. "To protect our lives, let us not fight with forks and frying pans," you suggest, "Let's use MATLAB."

Function Description:

You are given two structures which describe the stats of two corndog combatants. Each structure will contain exactly 5 fields which are described as follows:

- The field `'Name'` will store the fighter's name (*char*)
- The field `'Health'` will store the fighter's health (*double, positive integer*)
- The field `'Level'` will store the fighter's level (*double, positive integer*)
- The field whose name **contains the substring** `'Attack'` will store the fighter's attack stat (*double, positive integer*)
- The field whose name **contains the substring** `'Defense'` will store the fighter's defense stat (*double, positive integer*)

It is your job to use the information in each fighter's structure to simulate a battle and determine which of them wins. Fighters will take turns attacking each other, so in each turn one fighter is the attacker and one fighter is the defender. Fighter A (input 1) should attack first followed by Fighter B (input 2). Every turn, the attacker will deal damage to the defender based on the following formula:

$$Damage = \left(\frac{A_A}{D_D}\right) * L_A^2$$

Where A_A is the attacker's attack stat, L_A is the attacker's level, and D_D is the defender's defense stat. Once calculated, the damage value should be **rounded to the nearest integer**. After each attack, update the defender's health by subtracting the damage from it. Keep repeating this procedure, alternating who attacks and who defends, until one fighter's health reaches zero or lower. At that point you must end the battle and update the loser's structure by assigning their health to zero.

Your first output should be a string that describes the outcome of the battle, structured as follows:

```
'<Winner's name> destroyed <Loser's name> to become the
                2021 Corndog Champion!'
```

Your second output should be the updated structure of fighter A's stats, and your third output should be the updated structure of fighter B's stats.

Example:

```
fighterA =                                  fighterB =

          Name: 'Broccolee'                           Name: 'Coleeflower'
        Health: 135                                 Health: 110
  VeggieAttack: 60                          DefensePower: 30
   CornDefense: 50                            FireAttack: 70
         Level: 4                                   Level: 5

     [result, updatedA, updatedB] = corndogWars(fighterA, fighterB);

     result →
          'Broccolee destroyed Coleeflower to become the 2021
          Corndog Champion!'

updatedA →                                  updatedB →

          Name: 'Broccolee'                           Name: 'Coleeflower'
        Health: 30                                  Health: 0
  VeggieAttack: 60                          DefensePower: 30
   CornDefense: 50                            FireAttack: 70
         Level: 4                                   Level: 5
```

Notes:

- You are guaranteed both structures will never share an attack or defense field name.
- The words `'Attack'` and `'Defense'`, will appear as part of a fieldname exactly as written here, case sensitive.
- There will be only one fieldname containing the word `'Attack'` and one fieldname containing the word `'Defense'` in each structure.
- You are guaranteed positive integers for all fields that contain numbers.
- You are guaranteed both fighters will never have the same name.
- All battles are guaranteed to end eventually (i.e. no battles will have both fighters dealing 0 damage).

Hints:

- Break this problem down into its fundamental steps and consider the order of operations.
- Remember: order of fields in structures does not matter.