Function Name: `frankenCumber`

Inputs:

1. (*cell*) 1XN cell vector of characteristics and values

Outputs:

1. (*structure*) 1X1 structure with N fields of field names of characteristics and field values of the values

Topics: (*structure creation*)

Background:

You are a sea cucumber scientist who wishes to characterize all the sea cucumbers in the ocean.

Function Description:

You are given a 1XN cell array of cells with characteristics and values. In each of the cells in the cell array, the first index will be the characteristic and the second index will be the value. Create a 1X1 structure with field names as the characteristics and the values as values of the structure.

Example:

```
cellData={{'Length',5},{'Height',4},{'hasEyes',true},{'Color','Red'}}
structData=frankenCumber(cellData)
structData →

 Length: | 5     |
 Height: | 4     |
hasEyes: | true  |
  Color: | 'Red' |
```

Function Name: `cucumberCollection`

Inputs:

1. (*struct*) 1xN vector of structures of sea cucumbers you have

Outputs:

1. (*struct*) 1xM vector of structures of sea cucumbers to be released

Topics: (*structure masking*), (*field deletion*)

Background:

You are a sea cucumber specialist and your collection of sea cucumbers is the largest in the world. It is becoming difficult to manage your collection of sea cucumbers, and so you decide to release some sea cucumbers into the sea. But you only want to release the stronger sea cucumbers because they have a higher chance of survival in the wild.

Function Description:

Given a vector of structures containing the fields `HP`, `Regeneration`, `Stealth` `Mobility`, and `Intelligence` where each structure represents a sea cucumber, perform the following steps to output a vector of structures representing the sea cucumbers that are fit to be released.

1. Add a new field called `XP` to the structures in the vector that contains the XP values calculated using the formula:

   ```
   XP = 10*HP + 5*(Intelligence + Regeneration) + Stealth
   ```

2. Find and keep only the sea cucumbers that can be released. These sea cucumbers must have **at least one** of the following characteristics:
   - An XP over 1000
   - A Stealth over 70
3. Since sea cucumbers do not have the best mobility, remove the `'Mobility'` field from the chosen sea cucumbers as it is not important.
4. Finally, sort the sea cucumbers based on the XP field in **descending** order.

(continued...)

Example:

```
collection →
```

HP:	16	60	45	82	10	86
Regeneration:	79	26	8	53	96	8
Intelligence:	31	65	22	99	0	39
Mobility:	9	12	16	1	13	4
Stealth:	16	74	15	44	81	80

```
selected = cucumberCollection( collection )
selected →
```

HP:	83	86	86	10
Regeneration:	53	8	8	96
Intelligence:	99	39	39	0
Stealth:	44	80	80	81
XP:	1624	1175	1175	661

Notes:

- You're always guaranteed to have at least one sea cucumber that can be released.
- All values in the structure will be of type double.

Function Name: `lostAtSeaCucumber`

Inputs:
1. (*struct*) A 1xN vector of structures containing info about your current and next location
2. (*char*) The name of the sea cucumber you are looking for

Outputs:
1. (*char*) A vector representing the names of the sea cucumbers you talked to in your search for the sea cucumber

Topics: (*dot operator*)

Background:

You are a lost sea cucumber who got separated from your S.S.C. (Significant Sea Cucumber) at the annual sea cucumber appreciation festival. When you ask a nearby sea cucumber where they went, they point you to the last sea cucumber spotted with your S.S.C. You ask this sea cucumber, and they point you to yet another sea cucumber last spotted with your missing S.S.C. You will use MATLAB to go down the line of sea cucumbers until you find your dearest sea cucumber.

Function Description:

Starting at the first index of your structure vector, iterate through the structure vector to find the desired name within the field `Name`. If that value in `Name` does not match your desired name, retrieve the double value in `Next` and move to that index in your structure vector. Continue iterating through your structure vector until you either:

A. Reach the structure with your desired name in `Name`
B. Return to a structure that you have already been at before, as you are going in a circle

You will then output a char vector with all the values in `'Name'` that you iterated through (separated by one space after each name, including the last one). Your output will include the name of the last structure you moved to, even if it is previously listed in your vector.

Example:

`seaCucumberCrowd ->`

`Name:`	`'Brenda'`	`'Daniel'`	`'Phyllis'`	`'Paul'`	`'Gertrude'`
`Next:`	`2`	`5`	`4`	`1`	`3`

```
searchingFor = 'Phyllis'
[searched] = lostAtSeaCucumber(seaCucumberCrowd, searchingFor)
searched -> 'Brenda Daniel Gertrude Phyllis '
```

Notes:
- Compare structures individually with `isequal()`, there may be two cucumbers named `'Bob'` who point to different other cucumbers. The final cucumber name will be unique.

Function Name: `findingSeaCucumber`

Inputs:

1. (*struct*) A 1xN struct vector representing a section of the ocean

Outputs:

1. (*double*) The distance away from shore
2. (*double*) The depth where the sea cucumber was located

Topics: (*dot operator*), (*nested structures*)

Background:

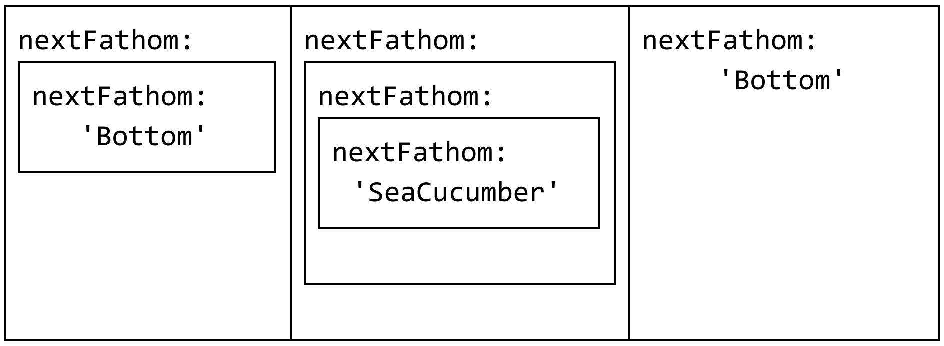
Your good friend Marlin has managed to lose his "son" during your school trip to Sydney, Australia. You cannot fathom why Martin calls his pet sea cucumber his son, but being the good friend (and MATLAB expert) that you are, you decide to help him find his "son."

Function Description:

Given a 1xN structure vector representing a section of the sea to search, find how far away from shore Marlin's pet sea cucumber was found, along with how deep it was. Each structure in the vector represents one unit of distance away from the shore, and each structure will only have 1 field: `nextFathom`. That field will have one of 3 values, `'SeaCucumber'`, `'Bottom'`, or another 1x1 structure that will only have the field `nextFathom`. Starting at the first section of the sea, keep searching deeper until you hit the bottom before moving on to the next section, repeating the process until you find the lost `'SeaCucumber'`. Each layer of nesting represents one unit of depth and each section of the sea represents 1 unit of distance.

Example:

`sea →`



```
[dist, depth] = findingSeaCucumber(sea);
dist → 2
depth → 3
```

Notes:

- There will always be a `'SeaCucumber'` present

Function Name: `theLastCucumber`

Inputs:
1. (*structure*) A structure representing the sea cucumber that is you
2. (*structure*) A structure representing the sea cucumber that is the opponent
3. (*cell array*) An Nx1 cell array of the moves (in sequential order) performed

Outputs:
1. (*cell array*) An Mx1 cell array of the move descriptions per turn

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

Background:

After having so many adventures with your fellow invertebrates, you decide to retire and live among the wild plains of the sea floor with your family! Farming one day for some plankton, you notice that your neighbor has pushed a pebble slightly too far into your turf. What the heck! You confront him about this, furious at his audacity and disrespect towards your land! The argument soon becomes heated after he calls you a dirty bottom feeder, the biggest insult any sea cucumber could ever receive. The nerve! Now, this isn't an argument, it's a battle: a battle to the last sea cucumber.

Function Description:

It's a battle until there's only one sea cucumber standing! This battle will alternate between the 'me' sea cucumber and its opponent, and it will always start from the 'me' sea cucumber. This means that your opponent will make the next move in the second round after the 'me' sea cucumber finished the first round, and so on.

The third input is the list that contains the move that one of the sea cucumbers uses. The moves are listed in the order of each round. Notice that the moves are not guaranteed to be unique for one sea cucumber. Here is the way that we are going to keep track of this battle:

- If you are attacking this round, add the following string onto your output cell array:

```
'You deal <numerical damage> damage to the opponent using the move <current move>!'
```

- If your opponent is attacking this round, add the following string onto your output cell array:

```
'The opponent deals <damage> damage to you using the move <current move>!'
```

The battle goes on only until there is one winner. When one of the sea cucumber's health number drops to 0 or below, that sea cucumber is the loser of the battle. The 'health' field is guaranteed to be given as a field of both sea cucumbers. The sea cucumber can win before you exhaust the move list at the third input. To find out who wins, we printed:

- If you are the winner and your opponent loses, add the following two separate lines onto the end of your output cell array:

```
       'The opponent suddenly faints, too exhausted to continue the battle!'
'You reign victorious with <your end health> health remaining! All beings over the
                              sea fear your might!'
```

- If your opponent is the winner and you lose, add the following line onto the end of your output cell array:

```
            'You suddenly faint, too exhausted to continue the battle!'
 'The opponent takes over your turf with <opponent end health> health remaining.'
```

Example:

```
me =                                opp =

   Tail_Attack: | 10 |                         Bounce: | 5  |
          Snap: | 20 |                  Spit_Plankton: | 10 |
        health: | 50 |                    Tail_Attack: | 7  |
                                               health: | 40 |

moves =
    {'Tail_Attack'  }
    {'Bounce'       }
    {'Snap'         }
    {'Spit_Plankton'}
    {'Tail_Attack'  }

>> out = theLastCucumber(me,opp,moves)
out =
  7×1 cell array
    {'You deal 10 damage to the opponent using the move Tail_Attack!'}
    {'The opponent deals 5 damage to you using the move Bounce!'}
    {'You deal 20 damage to the opponent using the move Snap!'}
    {'The opponent deals 10 damage to you using the move Spit_Plankton!'}
    {'You deal 10 damage to the opponent using the move Tail_Attack!'}
    {'The opponent suddenly faints, too exhausted to continue the battle!'}
    {'You reign victorious with 35 health remaining! All beings over the sea
fear your might!'}
```

Notes:

- The move list will always be longer or equal to the moves necessary.