Function Name: `halloweenParty`

Inputs:

1. (*cell*) 1xN cell array with information about your party

Outputs:

1. *(struct*) 1x1 structure with the fields and their values

Banned Functions:

`cell2struct`

Background:

You're trying to throw a well-*structured* Halloween party. All of those pesky party planning particulars prevent proper party planning, so you turn to structures to organize the data for your spook-tacular Halloween party.

Function Description:

Given a cell array of the format `{ field1, value1, field2, value2...}`, write a function that outputs a structure with each field name and the corresponding value for the field. Each field name is at the odd indices in the array and the respective value is at the adjacent even index.

Example:

```
>> ca = {'SpookFactor',1371,'IsLame',false,'Location','Culc 272'}
>> st = halloweenParty(ca)
>> st →
```

```
    SpookFactor: 1371
      IsLame: false
  Location: 'Culc 272'
```

Notes:

- The length of the input cell array will be even so that the values can be split up into field/value pairs.
- The values can be of any type, but the fields are guaranteed to be character vectors

Hints:

- Iteration might be useful to build the structure.

Function Name: `spookYourTA`

Inputs:

1. (*struct*) A 1xN structure array of all your TAs
2. (cell) A 2xM cell array with categories and their factors

Outputs:

1. *(char*) The name of the TA you want to SPOOK!

Background:

It's almost Halloween and it's the perfect time to get back your 1371 TAs for all the work they've put you through! You have developed an algorithm for deciding which TA you want to SPOOK!

Function Description:

You are given a structure array that describes the TAs of 1371. In each structure, the `Name` field will hold the name of the TA, and the rest of the fields are TA statistics, where the field name is the name of the statistic, and the value is the value associated with it.

The second input contains a cell array that holds names of statistics in the first row and their corresponding factor in the row below. For each statistic in the second input, multiply each TA's score by the factor for that statistic category. As you do this for each of the statistic, sum up the points for each TA. You want to spook the TA with the greatest number of points, so output the following sentence:

`'The highest score was <highest score> points. SPOOK <TA>!!!'`

Example:

```
>> TAs =
```

Name: Addison	Name: Dolly	Name: Justin
HW_Problems_Written: 5	HW_Problems_Written: 10	HW_Problems_Written: 50
Recitation_Students: 30	Recitation_Students: 35	Recitation_Students: 28
Help Desk Hours: 0	Help Desk Hours: 3	Help Desk Hours: 4
Average Test Score: 75	Average Test Score: 80	Average Test Score: 60

```
>> stats = [{'HW_Problems_Written'}, {'Average_Test Score'}; {10}, {-1}]
>> spookedTA = spookYourTA(TAs, stats)
>> spookedTA → 'The highest score was 440 points. SPOOK Justin!!!'
```

(Addison has a score of -25, Dolly has a score of 20, and Justin has a score of 440, so Justin is the TA to SPOOK!!!)

Notes:

- Each statistic in the cell array is guaranteed to be a field in the structure, but the structures may contain additional fields.
- There will never be a tie.

Function Name: `ghostBusters`

Inputs:

1. (*struct*) A 1x1 structure representing the containment unit

Outputs:

1. *(double*) The number of ghosts you freed

Background:

For years you have been trying stop the Ghostbusters and their unjust entrapment of ghosts across the city. You have broken into their headquarters and have reached the containment unit. It's finally your chance to set all of the ghosts free!

Function Description:

Write a function that frees all of the ghosts in the containment unit! Ghosts are stored at different levels within the containment unit to prevent them from helping each other escape. All of the levels are stored as nested structures, with the next level to check stored in the field `nextLevel`. The status of each level is stored in the field `ghost`, which is true if there is a ghost in that level of the containment unit. You must count the number of ghosts you are able to free from the unit, and output the number.

Example:

`Level1 →`

```
ghost: true
nextLevel: Level2
```

`Level2 →`

```
ghost: false
nextLevel: Level3
```

`Level3 →`

```
ghost: true
nextLevel: []
```

```
>> freed = ghostBusters(Level1)

out → 2
```

Notes:

- The `nextLevel` field of the last structure will be empty.

Function Name: `pumpkinPatch`

Inputs:

1. *(struct)* MxN pumpkin patch array

Outputs:

1. *(struct)* Best pumpkin

Background:

You are Linus van Pelt, Charlie Brown's neighbor. As Halloween approaches, you need to prepare for the arrival of the Great Pumpkin by finding the best pumpkin in the neighborhood pumpkin patch! Since you don't want to be disappointed again by the Great Pumpkin, you need to make sure you find the sp00kiest pumpkin in the pumpkin patch, so you turn to MATLAB to help you out!

Function Description:

You are given a MxN structure array, where each structure represents a pumpkin. The pumpkins will have several fields with numerical values, along with one field that contains text. Score each pumpkin by summing up all of the numerical values, and output the structure that has the highest total score. If a pumpkin contains the string `'Great Pumpkin'` in the field that contains text, output that pumpkin instead.

Example:

```
>> patch =
```

color: orange size: 5 weight: 30	color: not orange size: 10 weight: 35	color: orange size: 50 weight: 28

```
>> out = pumpkinPatch(patch)
>> out →
```

color: orange size: 50 weight: 28

Notes:

- Only one pumpkin will ever be the Great Pumpkin, and there will only be one pumpkin with the highest score.

Function Name: `costumeContest`

Inputs:

1. (*struct*) 1xN structure array with the description of a person's costume
2. (*char*) The banned field

Outputs:

1. *(char*) A phrase stating who the winner is and who the runner up is
2. (*struct*) 1x(N-1) structure array of the losers of the contest

Background:

It's officially *sp00ky* season. You've been planning your costume all year long and now it's time to show this whole town what you've got. But alas! The judges say it'll take all night to finish the scoring by hand and you were planning to go get your trick or treat on right after the contest. So, to help the judges, you create a function in MATLAB to speed up the process.

Function Description:

Write a function that takes in a 1xN structure array of contestants and finds the winner based on these criteria. Each structure will contain a `Contestants` field with the name of the person, an `Originality` field with a numerical score, and a series of fields that represent attributes of their costumes.The values of those fields will be 'Yes' or 'No'.

1. Delete the contestants with a value of `'Yes'` for the banned field, then delete the field from the structure array.
2. Find if the Cardboard and Makeup fields exists. Then find the contestants that use `Cardboard` or `Makeup`. Add three points to their originality if they have a `'Yes'` as the value for either of those fields. If they have both `Cardboard` and `Makeup` add 8 points to their originality score.
3. Sort the structure array by the `Originality` score in descending order.
4. Remove the `Originality` field to keep the scores a secret.
5. Find the first place winner (the costume with the best originality score) and the runner up and output the results like so:

```
'<Winner> is the winner of the costume contest! and <Runner-up> is the first
        loser, I mean second place winner of the costume contest!'
```

6. Output the losers in a structure array. This is everyone except first place.

Continued...

Example:

```
>>banned = 'Jewelry'
>>costumes =
```

Contestants: Buzz Tights: Yes Cardboard: Yes Jewelry: No Originality: 10	Contestants: Danny Tights: No Cardboard: Yes Jewelry: No Originality: 5	Contestants: Patricia Tights: Yes Cardboard: Yes Jewelry: Yes Originality: 7

```
>> [results, losers] = costumeContest (costumes, banned)
>>results → 'Buzz is the winner of the costume contest! And Danny is
              the first loser, I mean second place winner of the costume
           contest!'
>>losers →
```

Contestants: Danny Cardboard: Yes Tights: No

Notes:

- The disqualifying feature will always be a field name.

Hints:

- Remember that you can mask cell arrays.

Extra Credit

Function Name: `trickOrTreat`

Inputs:

1. *(struct)* 1XN Vector of structures representing houses
2. *(cell)* Mx2 cell array of candy preferences
3. *(double)* Number of houses you can visit

Outputs:

1. *(cell)* Px2 cell array of candy you have at the end of the night
2. *(cell)* Order of houses which you visited

Background:

After you have your sp00ky costume and completed your sp00ky CS 1371 homework, it's time for the most important part of Halloween: candy! Your neighborhood has some great trick or treating, but you can't go to every house because you need to wake up early to go to CS 1371 lecture! Naturally, you turn to MATLAB to help you plan out your Halloween night.

Function Description:

You are given a vector of structures where each structure represents a single house. The structure will have a `'Name'` field, containing the name of the family in the house, a `'Count'` field, containing the number of candy you can get from that house, and a `'Candy'` field, containing a cell array of the types of candy that house has. You are also provided a cell array with the names of candies you like in the first column and the corresponding numerical rating value for each candy in the second column. Determine the total score of every house by multiplying the number of candy you can get by the rating of your favorite candy you can get from that house.

You then need to determine the order in which you will visit the houses. You will always start at the middle house, your own, and get candy from your own house. Store the name of the candy that you got from each house into the first column of a cell array, and the amount of candy you got into the second column.

Then, you will either go to the house on the left or on the right, choosing the one with the higher score. Store how much candy you got at that house by either adding the name and count to the bottom of the cell array if you have not already gotten that type of candy, or by adding the number of that type candy you just got to the number of that type of candy you already have.

After that house, you will go to either the closest house on the left of the house you are at that you have not already visited, or the closest house on the right in the vector that you have not already visited. If you are all the way to the right or left, then you keep going in the other direction. If they have the same score, go to the house in the direction of your own house (in the middle). Repeat the process of storing the candy you got for each house. Output a cell array of the names of the houses you visited in the order you visited them and the candies cell array.

Example:

`houses =`

Name	'Htay'	'Ho'	'My house'	'Schmitt'	'Profili'
Count	6	6	9	2	2
Candy	{'Twix', 'Skittles', 'Kit Kat'}	{'Mochi', 'Twix', 'Skittles'}	{'Mochi', 'Kit Kat'}	{'Kit Kat', 'Pocky', 'Mochi'}	{'Mochi', 'Skittles'}

`candy =`

'Twix'	4
'Pocky'	1
'Skittles'	3
'Mochi'	5
'Kit Kat'	2

```
[bag,path] = trickOrTreat(houses,candy,4)
bag = {'Mochi',17;'Twix',6}
path = {'My House','Ho','Htay','Schmitt'}
```

First, calculate the scores of all the houses, which are `[6 * 4, 6 * 5, 9 * 5, 2 * 5, 2 * 5] = [24,30,45,10,10]`. We then start at `My house`, since that is in the middle. We add 9 Mochi to our bag. We then compare the Ho house to the Schmitt house, since they are to the left and right, and realize the Ho house has a higher score. We then move to the Ho house and add 6 Mochi to our bag. We then compare the Htay house to the Schmitt house and realize the Htay house has a higher score. We then move to the Htay house and add 6 Twix to our bag. Since there are no houses to the left, we move to the Schmitt house on the right, and then add 2 Mochi. That was the fourth house, so we stop there.

Notes:

- Every candy has a unique rating value.
- There will always be an odd number of houses.
- At each house, you only have to consider at most two houses for the next house you visit.
- You are guaranteed to have a rating for every candy you can find.
- The number of houses you can visit includes visiting your own.
- The names of the houses will be unique.