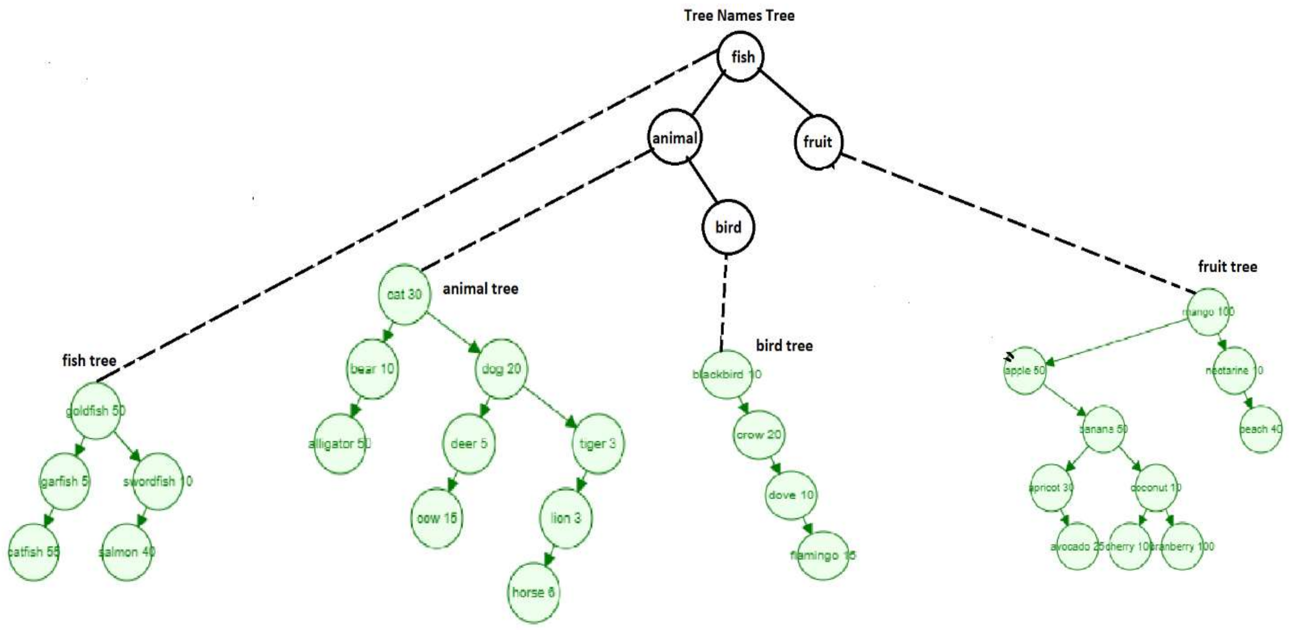
Problem: Almost a Forest

In this assignment you will build many BST trees where each tree has a name. To store the names of all the trees, you will maintain a binary search tree for the tree names.

After building the trees, you will have to perform a set of operations and queries.

Here is an example. In this example fish, animal, bird, and fruit are part of the name BST. Each node of the name tree points to a BST of items. Here, the fish node points to a BST that contains name and count of fishes. Note that all the green colored nodes are of same type of node structure and all the black colored nodes are of same type of node structure.



Input Specification:

You have to read the inputs from **in.txt** file. There will many strings in this file and you can assume that all the strings will be lower case letter and the maximum length of a string is 30 character. For simplicity, use static array of char to store a string.

The first line of the file contains three integers N, I, and Q, where N represents number of Tree Names, *I* represents the total number of items in the list to be inserted to all the trees, and Q represents the number of queries listed in the input file.

After the first line, next N lines will contain the list of Names of the trees and you need to insert them to a Name tree.

Next, I lines will contain the list of items to be inserted in different trees. Each line of the list contains two strings and one integer. The first string contains the name of that tree, second string contains the item name, and then the last integer contains the count of that item. You have to insert the item in the tree with that tree name. You need to use the item name as the key of the BST. Also note that the item count need to be added to the node as well.
[Assumption: You can assume that a tree name as well as an item name will not be repeated in the input]

After the *I* lines, the next Q lines will contain a set of queries and you need to process them.
Here is the list of queries:

- search: search for a particular item in a given tree and display the count of the item if it is found. Otherwise, prints item not found. However, if the tree does not exist, then it prints tree does not exist
  - [Example: *search fruit avocado* should search for avocado in the fruit tree and then prints the count of avocado. If the fruit tree exists, but the avocado does not exist, it should print "not found". However, if fruit tree does not exist, then it should print tree does not exist.
- item_before: this command counts the items in a given tree coming before a given item name (alphabetically).
  - [For example, if your query is like this *item_before animal deer,* it should print 4 as cat, bear, alligator, and cow come before deer alphabetically.]
- height_balance: It finds whether a given tree is height balanced or not. In order to do that, you need to know the height of left **sub tree** and then height of right **sub tree**. If their height **difference's absolute value** is more than 1, then we say the tree is imbalanced. In this assignment, a tree with 1 node, will be consider as height 0, and a tree with no node will be considered as height -1.
  - [Example: *height_balance animal,* will print, left height 1, right height 3, difference 2, not balanced]
- count: this command prints the total number of items in a given tree.
  - [Example: *count animal,* should print 142 (as 30 + 20+ 10 +50 +3 + 3 +5 +15 = 142)]
- reduce: this command reduces the count of an item in a given tree. The min value of the count needs to be >0. So, if it becomes <=0, you need to delte the node] //This command is optional. If you implement it, you will get **5 points extra credit**. So, keep it aside to do later if you get time.
  - [Example: *reduce fruit mango* 50 will reduce the total mango to 50 from 100.]

    - [Another Example: *reduce goldfish 50* will delete the node as goldfish is reduced to 0]
- delete: this command deletes an item from a given tree.
    - [Example: *delete fruit avocado* will delete the avocado node from the fruit tree]
- delete_name: this command delte the entire tree of a given name.
    - [Example: *delete_name animal* will delete the animal tree as well as animal node from the name tree]

Sample Input:

```
4 28 21
fish
animal
bird
fruit
animal cat 30
fish goldfish 50
animal dog 20
bird blackbird 10
animal bear 10
fruit mango 100
animal alligator 50
animal tiger 3
animal lion 3
fish swordfish 10
animal deer 5
animal cow 15
fish garfish 5
fish catfish 55
fish salmon 40
bird crow 20
bird dove 10
bird flamingo 15
fruit apple 50
fruit banana 50
fruit nectarine 10
fruit coconut 10
fruit peach 40
fruit apricot 30
fruit avocado 25
fruit cherry 100
fruit cranberry 100
```

```
animal horse 6
search fruit avocado
search fish tilapia
search animal cow
search bird crow
search bird cow
search animal cat
item_before animal deer
height_balance animal
height_balance bird
height_balance fish
search flower rose
count animal
count fruit
delete animal cat
search animal cat
count animal
delete fish swordfish
delete fruit avocado
delete_name animal
reduce fruit mango 50
search fruit mango
```

Output Specification:
You have to write all the output to an **out.txt** file. You are allowed to use a global variable for outfile pointer to simplify your function parameters.
After building the tree, you should print the trees in inorder in the specified format shown in the sample output bellow.

Sample Output:

```
animal bird fish fruit
===animal===
alligator bear cat cow deer dog horse lion tiger
===bird===
blackbird crow dove flamingo
===fish===
catfish garfish goldfish salmon swordfish
===fruit===
apple apricot avocado banana cherry coconut cranberry
mango nectarine peach
25 avocado found in fruit
tilapia not found in fish
```

```
15 cow found in animal
20 crow found in bird
cow not found in bird
30 cat found in animal
item before deer: 4
animal: left height 1, right height 3, difference 2, not balanced
bird: left height -1, right height 2, difference 3, not balanced
fish: left height 1, right height 1, difference 0, balanced
flower does not exist
animal count 142
fruit count 515
cat deleted from animal
cat not found in animal
animal count 112
swordfish deleted from fish
avocado deleted from fruit
animal deleted
mango reduced
50 mango found in fruit
```

Implementation Restriction.:

1. You have to use the following structure. You are allowed to modify the structure if needed.

2.
   ```
   typedef struct itemNode
   {
     char name[MAXLEN];
     int count;
     struct itemNode *left, *right;
   }itemNode;

   typedef struct treeNameNode
   {
     char treeName[MAXLEN];
     struct treeNameNode *left, *right;
     itemNode *theTree;
   }treeNameNode;
   ```

2.In addition to typical functions of tree implementation, you must have to implement the following functions:

i. `createTreeNameNode()`

ii. treeNameNode* `buildNameTree(…..)`: Based on the data in the file, it will insert them to the name tree and then finally return the root of the name tree

iii. `traverse_in_traverse(treeNameNode *root):` this function takes the root of the name tree and prints the data of the name tree and the corresponding item trees in the format shown in the sample output. You can call other function from this function as needed.

iv. `treeNameNode * searchNameNode(treeNameNode * root, char treeName[50]):`
This function takes a name string and search this name in the name tree and returns that node.
This function will help you a lot to go to the item tree.

v. Note that you might need to crate separate insertion and other functions for name tree and item tree. [I mean only two insertion function.]

vi. All the numbers and output must be produced from the trees.

Hints:

- Always start as soon as possible as they might take time and you will face various issues during this process.
- Read the complete instruction first. It is always a good idea to plan it and do some paper work to understand the problem.
- Analyze sample input output and match them with the description and the tree.
- You can use the uploaded BST code, and practice problem codes. But you will have to significantly modify them
- Just start by building the name tree first and see the inorder traversal of it
- Then gradually build the other trees and test them as you go.
- Create functions to simplify your code and it will be easier to test your code, disable part of your code, etc.
- **Use discussion board for any question, so that others also get benefited from your question and answer.**
- Keep patience and take help from us when you get stuck :)

Your code must compile in EUSTIS server. If it does not compile in Eustis server, we conclude that your code does not compile even if it works in your computer.

Steps to check your output AUTOMATICALLY in Eustis or repl.it or other command line based compiler:
You can run the following commands to check whether your output is exactly matching with the sample output or not.
Step1: Copy the sample output into sample_out.txt file and move it to the server (you can make your own sample_out.txt file)
Step2: compile and run your code using typical gcc and other commands. Your code should produce out.txt file.
Step3: Run the following command to compare your out.txt file with the sample output file

```
$diff -i out.txt sample_out.txt
```

The command will not produce any output if the files contain exactly same data. Otherwise, it will tell you the lines with mismatches.
Incase if your code does not match, you can use the following command to see the result in side by side:

```
$diff -y out.txt sample_out.txt
```

Tentative Rubric (subject to change):

- If the code does not compile in Eustis server, it can get zero. There may or may not be any partial credit. However, if the code does not complete, it will not get more than 35% even if it has all the necessary codes.
- Building the name tree: 5%
- Building the other trees: 10%
- Traverse_in_traverse: 5%
- search command: 10%
- item_before command 10%
- height_balance command: 10%
- count command: 10%
- delete command: 10%
- delete_name command: 10%
- Passing test cases perfectly exact match: 20%
- Properly implementing the reduce command (+5 extra credit)
- Note that all command has to match the output format to get full credit in that particular command.

Penalty:

- Not freeing up memory (5%) and not using memory leak detector (-10%)
- Not writing required function (-20%)
- Badly indented code (-15%) and not putting comments in important places (5%)
- Hard coding any number or data should receive -150%