Project 2: Binary Search Trees (100 points)

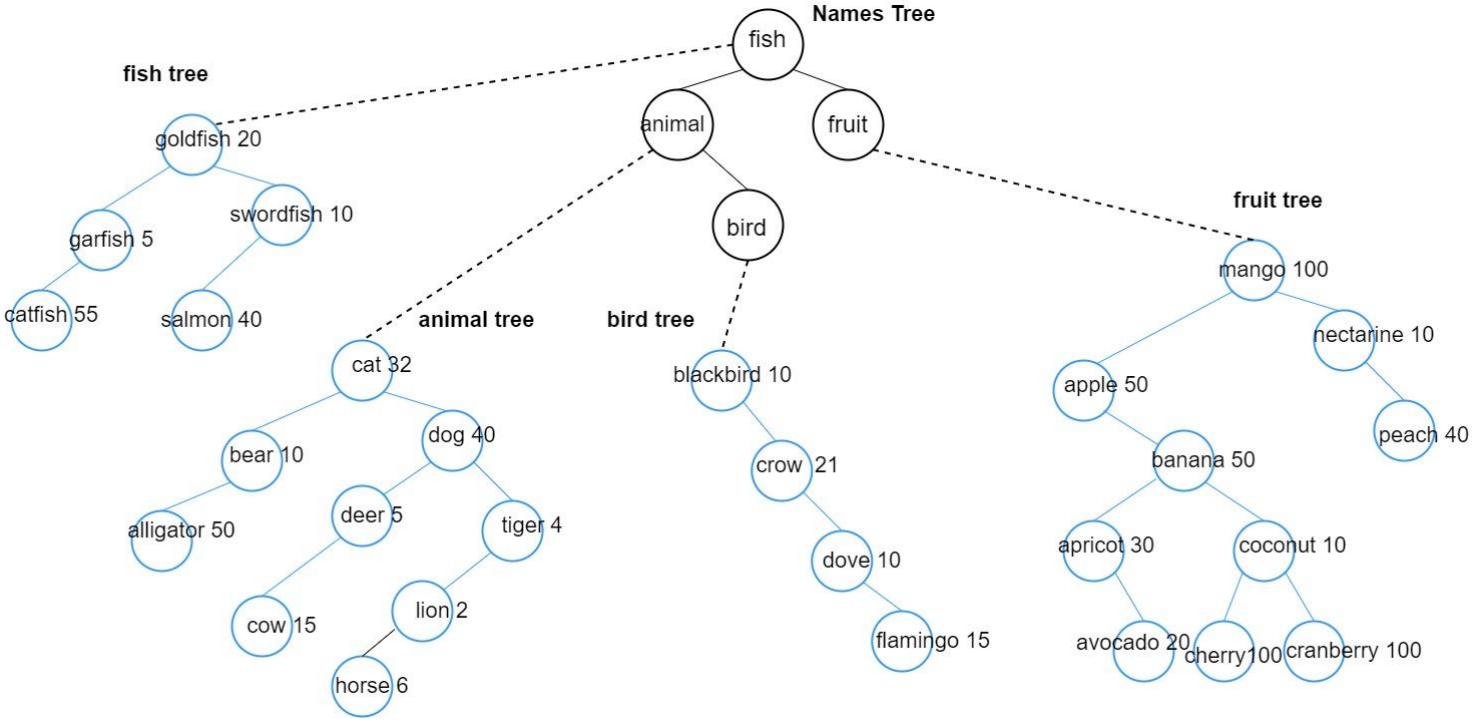
This project requires one cpp file submission. Do not upload a code snippet, the only files you submit for the code should be the source files. Do not submit any zip or compressed files, binary files, or any other generated files. Do not put the code inside the PDF file. Submission of unnecessary files or binary files in addition to source files will make you lose the points of the assignment.

The code must have the following:
a. Use consistent indentation through your code.
b. Use consistent braces style through your code.
c. File-level comments, these comments will include the file name, your name, course number, and date of last modification.
d. Function level comments include the function's description, the function's parameters, and the function's return value. These comments will appear before each of the functions, not the prototypes.
e. In function comments, these comments will include a description of the atomic functionalities within the bodies of the functions.
f. We evaluate your code using the following C++ online compiler. You will not get points if your assignment does not compile with this.

https://www.onlinegdb.com/online_c++_compiler

In this project, you will build many Binary Search 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 blue colored nodes are of same type of node structure and all the black colored nodes are of same type of node structure (same class objects).



You have to read the inputs from in.txt file. There will be many strings in this file and you can assume that all the strings will be lower case letters and the maximum length of a string is 30 characters. For simplicity, use a 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 needs 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 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).
  - 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 considered as height 1, and a tree with no node will be considered as height 0.
  - Example: height_balance animal, will print, left height 2, right height 4, difference 2, not balanced

- count: this command prints the total number of items in a given tree.
  - Example: count animal, should print 164 (as 32 + 40+ 10 +50 +4 + 2 +5 +15 + 6= 164)

Input example:

```
4 28 17
fish
animal
bird
fruit
animal cat 32
fish goldfish 20
animal dog 40
bird blackbird 10
animal bear 10
fruit mango 100
animal alligator 50
animal tiger 4
animal lion 2
fish swordfish 10
animal deer 5
animal cow 15
fish garfish 5
fish catfish 55
fish salmon 40
bird crow 21
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 20
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
search animal koala
search animal cat
count animal
search fruit mango
```

Output Specification:
You have to write all the output to an out.txt file as well as to the console. You are allowed to use a global variable for outfile pointer to simplify your function parameters. (FILE *outfile; can be at the top of your code)
After building the tree, you should print the trees in inorder traversal in the specified format shown in the sample output below.

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
20 avocado found in fruit
tilapia not found in fish
15 cow found in animal
21 crow found in bird
cow not found in bird
32 cat found in animal
item before deer: 4
animal: left height 2, right height 4, difference 2, not balanced
bird: left height 0, right height 3, difference 3, not balanced
fish: left height 2, right height 2, difference 0, balanced
flower does not exist
animal count 164
fruit count 510
koala not found in animal
32 cat found in animal
animal count 164
100 mango found in fruit
```

Implementation Restrictions::

1. You must use the following classes. You are allowed to modify them if needed

```
//For the items of each tree Name
class itemNode
{
  public:
         char name[50];
         int count;
         itemNode *left, *right;
         itemNode()
       {
         name[0]='\0';
         count = 0;
         left = NULL;
         right = NULL;

       }
        itemNode(char itemName[], int population)
       {
         strcpy(name, itemName);
         count = population;
         left = NULL;
         right = NULL;

       }
};

//For Tree Names
class treeNameNode
{
  public:
         char treeName[50];
         treeNameNode *left, *right;
         itemNode *theTree;
        treeNameNode()
       {
         treeName[0]='\0';
         theTree = NULL;
         left = NULL;
         right = NULL;
       }

       treeNameNode(char name[])
       {
       strcpy(treeName, name);
       theTree = NULL;
       left = NULL;
       right = NULL;
       }
};
```

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

`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

make another function for buildItemTree

`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 functions from this function as needed.

`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.
Note that you might need to create separate insertion (or other functions) for the Name tree and Item tree. (I mean only two insertion functions one for Name tree and one for Item tree)

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 and output and match them with the description and the tree.
- You can use the uploaded BST code, and other sample codes implemented in the class.
- Use `strcmp()` function for string comparison. (You need to #include <string.h>)
- Use `fscanf()` and `fprintf()` to read and write the files. (If you don't remember how to read strings and integers from a file and how to write into files you need to review those topics)
  EX: line 2 of txt file: fscanf(infile, "%s", charArray)
  ```
  FILE *infile = fopen("in.txt", "r");
  fscanf(infile, "%d %d %d", &treeNameCount, &itemCount, &queryCount);
  ```
  FILE *outfile = fopen("out.txt", "w")
  This line will read the first line of your input file and store the information in `treeNameCount`, `itemCount` and `queryCount`

- 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.

Tentative Rubric (subject to change):

- Building the Name tree: 15%
- Building the other trees(for items): 20%
- Traverse_in_traverse: 5%
- search command: 15%
- item_before command 10%
- height_balance command: 10%
- count command: 10%
- Passing test cases perfectly exact match: 15%
- Note that all command has to match the output format to get full credit in that particular command.