Programming Assignment 4: Almost a Forest

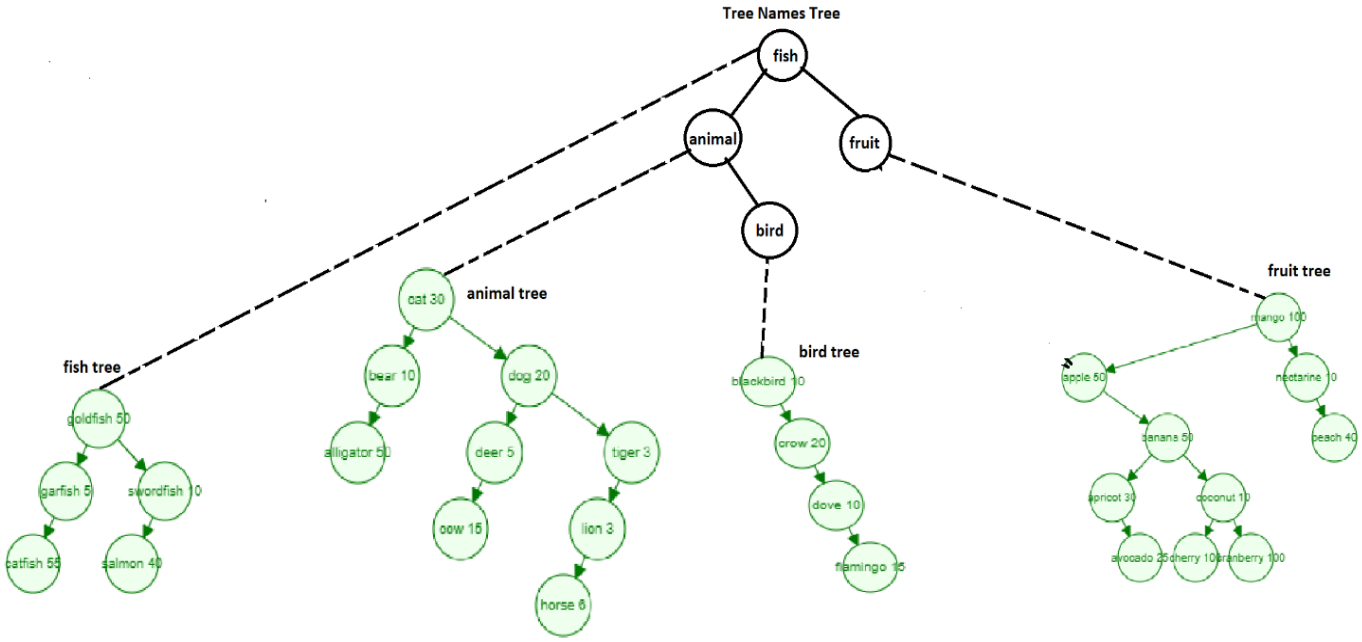
COP3502 – Spring 2021

Overview

When small monsters were first discovered, they were believed to be confined to forests. For years thereafter, regions where they could be captured were referred to as “forests”, regardless of their actual type of ecological region.

The mess this made of the historical record is a story for another time.

You are building a program to parse compressed records from that time period. Your program will maintain a set of trees, by maintaining a *tree of trees* to control those trees.



Data Structures

You *must* use the following structures for your trees. This is not optional, and exceptions will not be granted.

```
struct item_node_struct
{
  char name[32];
  int count;
  struct item_node_struct *left, *right;
};

typedef struct item_node_struct item_node;

struct tree_name_node_struct
{
  char treeName[32];
  struct tree_name_node_struct *left, *right;
  item_node *theTree;
};

typedef struct tree_name_node_struct tree_name_node;
```

The Input File

Restrictions

All strings will be in lowercase. The maximum length of any line is 63 characters. The maximum length of any name is 31 characters.

There will be neither duplicate tree names nor duplicate item names.

Structure

- The first line has three integer values n_{trees}, n_{items} and $n_{commands}$.
- After the first line, the next n_{trees} lines will be the names of the trees. These are the keys for the "top half" of the tree of trees.
- Following the names of the trees, the next n_{items} lines will contain two strings and an integer each:
  - s_{tree}, the first string, contains the name of the tree to insert an item in.
  - s_{item}, the second string, contains the name of the item. These are the keys for the "bottom half" of the tree of trees.
  - c_{item}, the integer, contains the item count.
- Following the items for insertion, the next $n_{commands}$ lines will contain a set of commands for your program to follow. Each command is one of the following:
  - **search <tree> <item>**
    - Search for item **item** in tree **tree**. Print the count if it's found, "<item> not found in <tree>" if the item doesn't exist there, or "<tree> does not exist" if the tree doesn't exist.
  - **item_before <tree> <item>**
    - Count the number of items lexographically before item **item** in tree **tree**.
  - **height_balance <tree>**
    - Prints the heights of the left and right subtrees of tree **tree**, and whether or not they're balanced.
    - A tree is balanced if the heights of its left and right subtrees differ by at most one.
    - A tree with only a root has height 0.
    - A null reference has height -1.
  - **count <tree>**
    - Prints the total number of items in tree **tree**.
    - Cackling is not permitted.
  - **delete <tree> <item>**
    - Deletes item **item** from tree **tree**.
  - **delete_tree <tree>**
    - Deletes tree **tree**.

Sample Input

```
4 28 19
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_tree animal
```

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
=====Processing Commands=====
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
```

Required Functions

You will need to create the following functions on top of the usual tree functions:

- search_for_name_node() to find a subtree in the “top half”.
- search_in_name_node() to find an item in the “bottom half” **given** a subtree in the “top half”.
- traverse_in_order() to print the tree in the format shown at the top of the output file.
- Separate insertion and deletion functions for the “top half” and “bottom half” of the tree.

Coding Requirements

- You do not need to comment line by line, but comment every function and every “paragraph” of code.
- You don’t have to hold to any particular indentation standard, but you must indent and you must do so consistently within your own code.
- You may not use global variables.

Submission and Naming

- Name your file **cop3502-as4-<yourlname>-<yourfname>.c**. For example, mine will be named **cop3502-as4-gerber-matthew.c**.
- Read input from **cop3502-as4-input.txt**.
- Write output to **cop3502-as4-output-<yourlname>-<yourfname>.txt**. For example, my output file will be named **cop3502-as4-output-gerber-matthew.txt**.