CPE 111 Programming with Data Structures
International Sections - January 2021
Laboratory Exercise 6

Objective

This lab is intended to give you the chance to create a simple sorted binary tree, to use as a search index.

Overview

Modify your program **linkCouples.c** from Lab 3. Create a new program called **treeCouples.c** that does exactly the same thing as the original program. However, the new program should store the people read from the input file in a sorted binary tree rather than in a linked list. This should make it faster to find a person, either to check whether that person is already in the data structure or to make the person part of a couple. Searching in a balanced sorted binary tree is **$O(\log_2 n)$** while searching a linked list is **$O(n)$.**

Details

- Start with your own **linkCouples.c**. You will find a link a directory of your Lab 3 solutions on the home page. If you did not submit Lab 3, you can use my code from the LabSolutions directory (but it would be better if you used your own).
- You will need to make the following changes:
    1. Modify the structure definition for **PERSON_T**. Instead of having a "next" pointer, you will need to have pointers to the left child and right child.
    2. Delete the head and tail pointers. Instead you will need a **PERSON_T*** variable for the root of the tree. Be sure to start by setting the root to NULL.
    3. Write a function to insert a **PERSON_T*** into the tree at the correct location in the tree, based on the value of the name, which is the key field for each node. To compare the name fields, use **strcmp()**. You might want to look at the **insertNode()** function in **binaryTree.c**, which is in the **demos** area for Lecture 2, or the **trythis6.c** example. Note inserting a node in a sorted tree requires a recursive function. Your function does not have to worry about duplicate keys, because you should call **findPerson()** to check if a name exists before adding it to the tree.
    4. Rewrite the **findPerson** function. This function should now recursively search the tree until it finds a match or has looked at every node. ***See the pseudocode below.***
    5. Change the code in **main()** that builds the linked list to call **insertNode()** instead. You will still need to allocate a **PERSON_T** and copy the name to that structure, as we did with **linkCouples.c**.
    6. Create a function called **printAll()**. This function should do an *in-order traversal* of the tree, printing the name at ever node. In **main** ask the user if she wants to print all the people. If the answer is yes, call **printAll()**. This code should replace the code that iterated through the linked list printing names.
    7. Rewrite the **printCouples()** function. Instead of iterating through the list, it should do an *in-order traversal* of the tree, printing the name and partner for any person who is part of a couple.
    8. Write a function to free the tree. This function should do a *post-order traversa*l, freeing the current node (PERSON_T) after going down the left and right branches. Call the **freeTree** function at the end of **main**, before you exit.

- You do not have to change the code that creates the connections between people, as long as you use a function to return pointers to the people to be matched.
- Test your program on **newpeople.txt** and **newpeople2.txt**. Compare it to the **linkCouples** program. Do you notice any difference in performance? If you want, you use my timing functions from Lecture 2 to examine this is more detail.
- Upload your C file **treeCouples.c**

Pseudocode for findPerson *(arguments - currentNode and name)*

```
set foundPerson to NULL
if currentNode is not NULL
    if name matches currentNode->name
        set foundPerson to currentNode
    else if name < currentNode->name
        foundPerson = findPerson(currentNode->leftChild)
    else
        foundPerson = findPerson(currentNode->rightChild)
endif /* not NULL */
return foundPerson
```

When you need to find a person, you will call this function by giving it the root node of your binary tree.