UCSC Silicon Valley Extension

C Programming, Advanced

Assignment 3 part 2: Data structures and graphs
Instructor : Radhika Grover

Include test cases in all of the following programs to demonstrate that the program executes correctly. The output carries 50% of the points for that problem. No points will be awarded if the program does not compile.

Linked list

1. Given two **sorted** linked lists, L1 and L2, write methods to compute :
    1. Intersection of the lists contains elements common to both lists: L1∩ L2
    2. Union of the lists L1 and L2 combines all the elements in both lists without duplicates : L1 ∪ L2

```
// Testcase 1
//list1 : 1, 3, 4, 8
//list2 :  2, 5
listUnion(list1, list2);  // 1, 2, 3, 4, 5, 8

// Testcase 2
//list1 : 1, 3, 4, 8
//list2 :  2, 3, 4, 5
listIntersection(list1, list2);  //3, 4

// Testcase 3
//list1 : 1, 2, 4, 5, 10, 12
//list2 : 1, 2, 3, 4, 5, 6, 7, 8
listIntersection(list1, list2); // 1, 2, 4, 5
listUnion(list1, list2);  // 1, 2, 3, 4, 5, 6, 7, 8, 10, 12


// Testcase 4
//list1 : 1, 2, 4, 5, 10, 12
//list2 : Empty List
listIntersection(list1, list2); //listIntersection returns empty list (list with no elements)
listUnion(list1, list2); // 1, 2, 4, 5, 10, 12
```

2. **Binary search tree** : create a binary search tree node that contains a data element (key) and two pointers to tree nodes called left and right. Write two functions to display the keys using postorder traversal:

(a) using recursion

(b) without using recursion (use a user stack instead)

```
// Testcase 1
add(tree, 5);
add(tree, 10);
add(tree, 3);
add(tree, 8);
add(tree, 20);
add(tree, 1);
post_order(tree); //1, 3, 8, 20, 10, 5


// Testcase 2
add(tree, 7);
add(tree, 8);
add(tree, 2);
add(tree, 8);
add(tree, 20);
add(tree, 1);
post_order(tree); //1, 2, 8, 20, 8, 7

// Testcase 3
add(tree, 7);
add(tree, 8);
add(tree, 2);
add(tree, 8);
add(tree, 20);
add(tree, 1);
add(tree, 11);
add(tree, 15);
post_order(tree); // 1, 2, 8, 15, 11, 20, 8, 7


// Testcase 4
add(tree, 11);
add(tree, 15);
post_order(tree); // 15, 11
```

Galactic search

3. [8 points] Race your classmates as you hitchhike your way through the galaxy in a spaceship powered by an infinite improbability drive. Only one obstacle remains in your quest to find the answer to the ultimate question of life, the universe and everything: you must first determine the nearest galactic neighbor, then visit it. Luckily, you have the right tool for it - a supercomputer named Deep Thought. Plug your program into this machine, and it may just yield the answer you're looking for.

For your convenience, the galactic coordinates have been converted to ordinary cartesian coordinates (x, y, z) and are stored in the file (Galaxies.txt) provided to you in the following format:

name of galaxy 1
x-coordinate, y-coordinate, z-coordinate
name of galaxy 2
x-coordinate, y-coordinate, z-coordinate

Write a program to read these records from the file into a data structure, and find the neighbor that is nearest your current coordinate of (x, y, z). To store the records, choose a data structure that is implemented using any **one** of the following: linked list, array, binary search tree. The data structure should support insertion and deletion of records. In addition, it should support an operation to print the record that is the nearest neighbor. Your program should read print out the total execution time of your program to print out the solution.

References

The Hitchhiker's Guide to the Galaxy:
https://en.wikipedia.org/wiki/The_Hitchhiker%27s_Guide_to_the_Galaxy