CS 305: HW4 Fast searches
Fall 2019

During the semester free tutoring is available to you for CS 305, please see tutors if you need help. If you have hard time in the class, e-mail the tutor fellow: stepup@up.edu for an appointment.

Due Date: NLT Sunday, November 10th 2018 at Midnight. Your code files (.h and .c files), makefile, 5 sample inputs that represent different test lists you used for testing your code. In your archive include answer questions in a pdf or docx file. Submit your HW as a single zip file to Moodle.

The key motivation for this assignment is the illustation of a tree datastructure to support the fast searches of the linked lists. A very similar implementation of trees is commonly used by relational databases to build the search indexes for fast information lookups.

You have two tasks to accomplish (1) is to create a linked list that stores airport information and (2) create two search trees that will reference the LList objects for fast lookup. The LList and struct structures are very similar to your previous homework, except they don't store printers/printJobs but flights. A couble observations:

- don't store the information in sorted order but in order the input file lists the airports
- there is exactly one instance of a airport struct, but multiple references from different trees
- a real life application would have as many search trees as there is "search by" terms

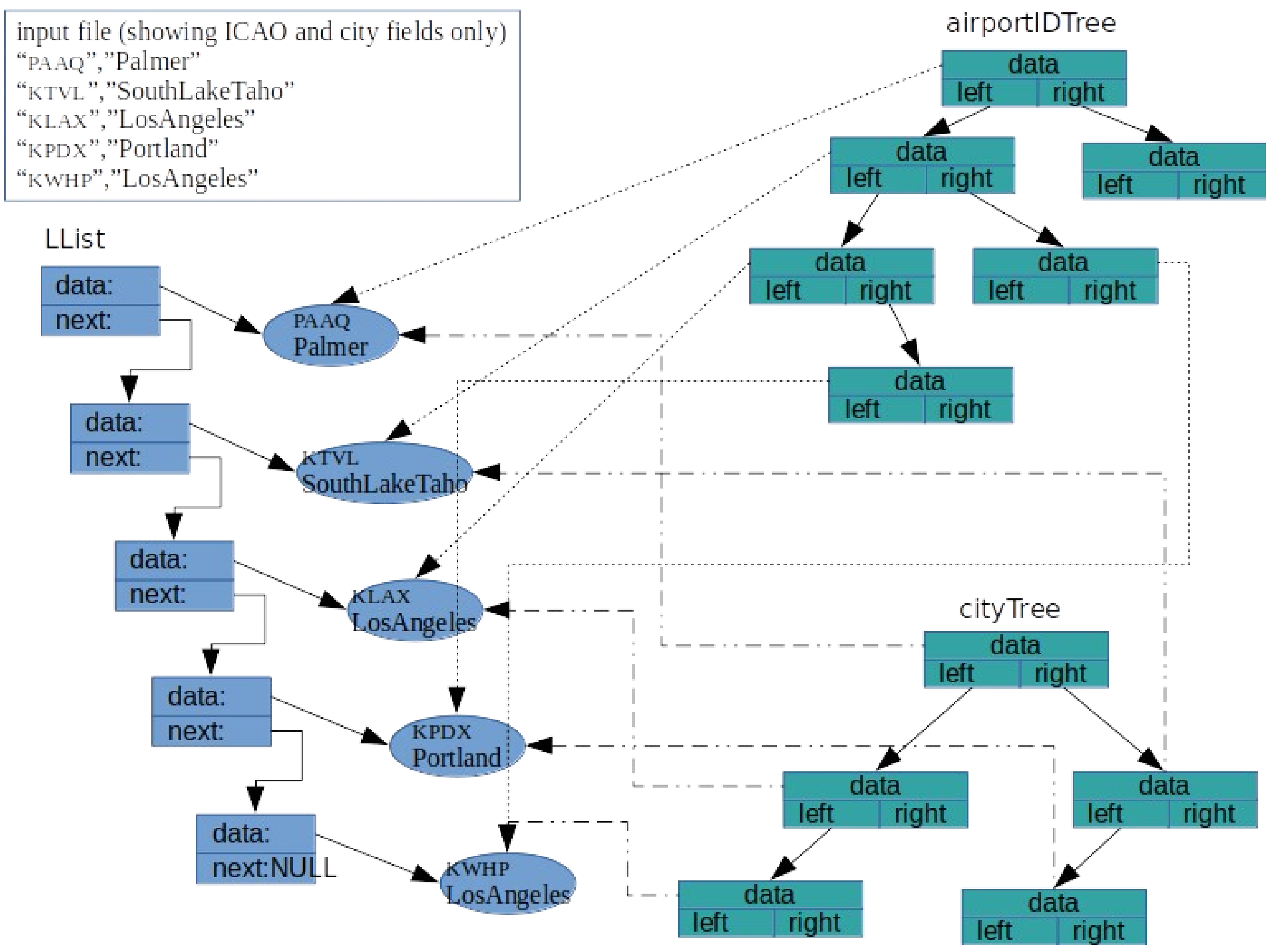
Your program fastLookup will read a text file as an argument, then executes a command line function.

The command line instructions include
./fastLookup <file name>
d <airportId>   # deletes the airport by airport ID from the trees and LList and the struct itself and repairs the structures as needed
sn <airportId>   # search for an ariport by airport ID in the id tree
sc <airportCity>   # search for all ariports in a search city in the city tree
sl <airportId>   # search for an ariport by airport ID in the LList
q   # quit the airport tracker and deallocate ALL memory

The airport struct data are stored in a struct called `airport`. The structs are stored in a LList by inserting at the head or tail.

The tree structure will build a binary search tree that will reference (point to) the customer structs stored in the LList and contain the rest of the airport information. The tree structure stores the pointer to the left and right child trees and the pointer to the airport data object. The search key is NOT stored in the tree node, but retrieved by following the pointer object and accessing the data object's field airportID or airport city. The functionality of the search function will simply print the entire airport record from the object or prints an error message. The search by city is tricky, as the corresponding search tree will have duplicate search keys and your search will have to descend all the way to the leaves to print out all matching ariports at the search city. The search by airport ID on the other hand can stop as soon as it find the search airport key since it is unique.

Motivating example abbreviated to use only airportID and city:



Operational requirements:
1. Airport ID data will be stored in a airport struct

2. A Linked List will keep track of the objects

3. All structures must be allocated on the heap and property freed when information is deleted or the application terminates

4. The linked list will support the following functions/operations:
    print LList – prints the list

search by airportID - find and print the airport details located by its unique airport ID.

5. The search trees will support the following functions/operations:
    search tree: by airport ID, or by city depending on which tree is used

6. Delete function will delete by airportID. This function will delete the entire record from LList, both search trees and the struct itself.
Implement delete simply and naively, find the struct in the LList, delete the LList node, but before deleting the struct, find the tree node in a tree, delete that node, repeat and find and delete the node from the other tree, and finally deallocate the airport node. (Note: The binary search trees as well as the LList must be property repaired after deletion)

7. I will let you design the rest of your program's functionality, structs, logic, functions. Couple suggestions: No function of more then 10-20 lines, each malloc has a matching free, separate the functionality operations on the LList, Tree, Airport ADT (don't mix these, unless you absolutely have to), ...

8. If you have time and/or want additional practice, you can add the command line functionality to add an airport <a> <airport information> which will add an airport as an object to a LList and adds the search nodes to both search trees.

9. Dealing with duplicates in the search by city BST. When you encounter an airport that is in the same city as another records (LosAngeles for example), insert the tree node at the time of BST creation immediately as the left child after the duplicate node. For example, if the root "Portland" has a left child "Palmer", when attempting to insert the second "Portland" into the city BST, it will be the left child of the first "Portland". The tree will look as "Portland" left child is "Portland" left child is "Palmer". Handling duplicates is required for city BST only. If the key is not duplicated, it is always added as a leaf to the tree.

10. The datafile has the following fields (and formats) that should be stored in your airport struct:

I.	AirportNumber	Unique number for this airport - irrelevant.
II.	Name	Name of airport.
III.	City	Main airport city. THIS IS YOUR SEARCH CRITERIA FIELD = city
IV.	Country	Country or territory where airport is located.
V.	IATA	3-letter IATA code. Null if not assigned/unknown.
VI.	ICAO	4-letter ICAO code. THIS IS YOUR SEARCH CRITERIA FIELD = airportID
VII.	Latitude	Decimal degrees. Negative is South, positive is North.
VIII.	Longitude	Decimal degrees. Negative is West, positive is East.
IX.	Altitude	In feet.
X.	Timezone	Hours offset from UTC.
XI.	DST	Daylight savings time. One of E (Europe), A (US/Canada), S (South America), O (Australia), Z (New Zealand), N (None) or U (Unknown).
XII.	Tz database time zone	Timezone in "tz" (Olson) format.
XIII.	Type	The airport type:"airport", "station", "port" and "unknown"

XIV. Source Source of this data. "OurAirports"

11. Yes, although we are only using two fields from each line, you have to parse and save the entire line in the airport struct.

Hints and Useful functions:

check out different formating for

strdup	- copy information from stack to heap
atoi	– convert a string to an integer
strtok	– tokenize a line
strcmp	- compares two strings against each other.
toupper	- converts character to upper case (included in ctype.h). It looks like to covert a string to upper case you have to do it character by character.

Design the delete function recursively, as you will need to set the previous tree node’s (left or right) pointer value to either null or a successor child.

Run your code through the valgrind to confirm you are not leaking memory.

On ‘q’ command line option, make sure you delete and deallocate the LList, airport structs, and both trees.

I don’t care if you add the airport to the LList at the head or tail.

Start with some super simple examples that will generate simple trees that you can easily check for correctness (by themselves, with the data pointers ignores) to make sure you can reliably delete tree nodes.

Grading criteria:

- (10 points) Code Operation: Does code do what is listed in the specification? Note: code that does NOT compile will receive 0 points for code operation.
    - 1 points print
    - 2 points make search tree
    - 3 points delete from search tree
    - 1 points malloc
    - 2 point free
    - 1 point makefile (works for compiling and cleaning)
- (5 points) Good programming practices including programming style
- (5 points) Summary report: Completeness, correctness, and clarity

HW 4 Report Guidelines and Format – use the template provided below. Include the questions in your write-up.
Name: **CS 305 HW 4 Report**

1. Questions (include these in your write-up):
1a. (.5 point) If your program does not meet the specifications, please note the differences and anything that is not working correctly. If it works, write “Meets all specifications.”

1b. (2 point) Copy and paste your terminal window inputs and outputs to show operation of your code. Generate 5 different data files using the original data file I provided for the homework. Show how the search times differ when using these files as an input to your program and searching them. To make the test files open the airport.dat file in a spreasheet program and sort the rows according the following criteria. Save the altered files as airport1.dat, ariport2.dat … 5. The alternatively sorted files should include the following:
file 1: all records sorted by airportID
file 2: all records reverse sorted by airportID
file 3: partially sorted records by city
file 4: partially reverse sorted records by city
file 5: randomly sorted records in the source data

Prepare a cumulative report of running searches on your files (a table would be nice). The searches should include looking for a record that does not exist, the last record inserted, record in the deepest branch of a tree, etc.

2c. (1.5 point) For each function in your program give a O (big-O) running time on input size n. This is not empirically measured, but calculated from the algorithm’s analysis. (Node, functions are named differently in your code)

Function	Best time	Worst time	Average time	What is the input that causes the best/ worst/ average running times?
Search airportID				
Search LList				
Delete airportID from all structures				
RepairTree (after deletion)				
Make airportID Tree				

3a. (.25 pt) How much time did you spend in total on this homework assignment (including the report)?

3b. (.25 pt) What was the most challenging part for you when completing this assignment?

Appendix A: (copy this statement if it applies to you) I verify that the code and this write-up were authored by me. I have documented the help I have received in comments in the code files.
Appendix B: Copy and paste your source files .c, your header files .h and makefile here (use Courier New 8pt font so the characters line up correctly)