COP 3502C Programming Assignment # 5
Binary Search Tree

Read all the pages before starting to write your code!

Objective:

To implement a binary search tree that handles strings with multiple fields, incorporating a range of functionalities for efficient data management and retrieval.

Deliverable and reward option as promised:

The bonus amount will be announced on webcourses.Write all the code in a single main.c file and upload the main.c file. Please include the following commented lines in the beginning of your code to declare your authorship of the code. Also, if you would like to be considered for the bonus, please add the following text on your comment as well and add a submission comment on webcourses with the same text before the bonus deadline,:

/* COP 3502C Assignment 5
This program is written by: Your Full Name */
/*I want to be considered for the bonus and I agree that any of my submissions after<date> (*put July 22 or July 23 based on your wish which level of bonus do you want*) will not be considered for grading regardless of my score on the most recent submission before <date> (put the same date here) */

[Also, make sure to add the same text on your submission comment on webcourses.]

Compliance with Rules: UCF Golden rules apply towards this assignment and submission. Assignment rules mentioned in syllabus, are also applied in this submission. The TA and Instructor can call any students for explaining any part of the code in order to better assess your authorship and for further clarification if needed.
Caution!!!
Sharing this assignment description (fully or partly) as well as your code (fully or partly) to anyone/anywhere is a violation of the policy. I may report to office of student conduct and an investigation can easily trace the student who shared/posted it. **Also, getting a part of code from anywhere other than class resources will be considered as cheating.**

Deadline:

See the deadline in Webcourses. An assignment submitted by email will not be graded and such emails will not be replied according to the course policy.

What to do if you need clarification on the problem?
Write an email to the TA and put the course teacher in the cc for clarification on the requirements. I will also create a discussion thread in webcourses, and I highly encourage you to ask your question in the discussion board. Maybe many students might have same question as you. Also, other students can reply, and you might get your answer faster.

How to get help if you are stuck?
According to the course policy, all the helps should be taken during office hours. There Occasionally, we might reply in email.

Smoothie Loyalty Program

Background Story

Our smoothie business is thriving! Thanks to our wide variety of recipes and the convenience of our store locations, we have become quite well-known in the area. We want to identify and track our most valuable customers, and introduce a loyalty program to reward their patronage and strengthen their relationship with our stores.

In this loyalty program, each customer earns 1 loyalty point for every dollar spent at our store. Over time, customers can accumulate loyalty points, redeem them for discounts, or check their current point balance. Additionally, customers have the option to opt out of the program and request removal from it. Since we are learning about binary search trees in class, we aim to implement this loyalty program using a binary search tree of nodes. These nodes will be compared based on the **customer's name**, sorted in alphabetical order. Furthermore, we would like to incorporate **debugging queries** to gain better insights into the tree structure as we develop this application.

Problem

Develop a program that processes input related to various updates and queries for the smoothie store's loyalty program. The program should output appropriate messages based on the input commands. Below is a detailed list of the potential commands that the program should handle:

1. *Add loyalty points to a specific customer.*
2. *Subtract loyalty points from a specific customer.*
3. *Delete a specific customer from the program.*
4. *Search for a specific customer in the binary search tree. If the customer is found, report both their number of loyalty points and their depth in the tree (measured as the number of links from the root).*
5. *Check the height of the left and right sides of the tree to determine if it is balanced relative to the root.*

Input (must be standard input (no file i/o is allowed))

The first line of input contains a single positive integers: ***n*** (***n*** ≤ 300,000), the number of commands to process.

The next ***n*** lines will each contain a single command. Here is the format of each of the possible input lines:

Command 1

`add <name> <points>`

`<name>` will be a lowercase alphabetic string with no more than 25 characters.
`<points>` will be a positive integer less than or equal to 100.

This command adds the customer with name <name> into the tree. If the customer is already in the system, it will increase the points of the customer with the given points.

Command 2

`sub <name> <points>`

`<name>` will be a lowercase alphabetic string with no more than 25 characters.
`<points>` will be a positive integer less than or equal to 100.
Note: if a customer has fewer points than is specified in this command to subtract, then just subtract the total number of points they have instead.

Command 3

`del <name>`

`<name>` will be a lowercase alphabetic string with no more than 25 characters.
Delete the customer with the name `<name>` from the binary search tree. No action is taken if the customer isn't in the tree.

Command 4

`search <name>`

`<name>` will be a lowercase alphabetic string with no more than 25 characters.

This will search for the customer with the name `<name>` and report both the number of loyalty points the customer has and the depth of the node in the tree storing that customer, if the customer is in the tree.

Command 5

`height_balance`

This will calculate the height of the left sub tree (It means height of the tree starting from the left node of the root), and heigh of the right subtree. If both heights are same, it should show that they are balanced. Otherwise, it should show the height is not balanced.

Output (standard output. No file i/o allowed)

For each input command, output a single line as described below:

Commands 1 and 2

Print out a single line with the format:

`<name> <points> <depth>`

where `<name>` is the name of the customer who added or subtracted points, `<points>` is the new total number of points they have, and `<depth>` is the depth of the node containing the name. If the customer in question is not in the tree while performing the sub command, then print:

`<name> not found`

Command 3

If the customer in question wasn't found in the binary search tree, output the following line:

`<name> not found`

If the name is found, output a line with the following format:

`<name> deleted`

where `<name>` is the name of the customer being deleted. (Of course, delete the node storing that customer from the tree!) **If you are deleting a node with two children, please replace it with the maximum node in the left subtree. This is to ensure there is one right answer for each test case.**

Command 4

If the customer in question wasn't found in the binary search tree, output the following line:

`<name> not found`

If the name is found, output a line with the following format:

`<name> <points> <depth>`

where `<name>` is the name of the customer being searched, `<points>` is the number of loyalty points they currently have and `<depth>` is the distance of the node the customer in question was found in from the root node of the tree.

Command 5

This command prints a line with the left height, right height and a message whether it is balanced or not.

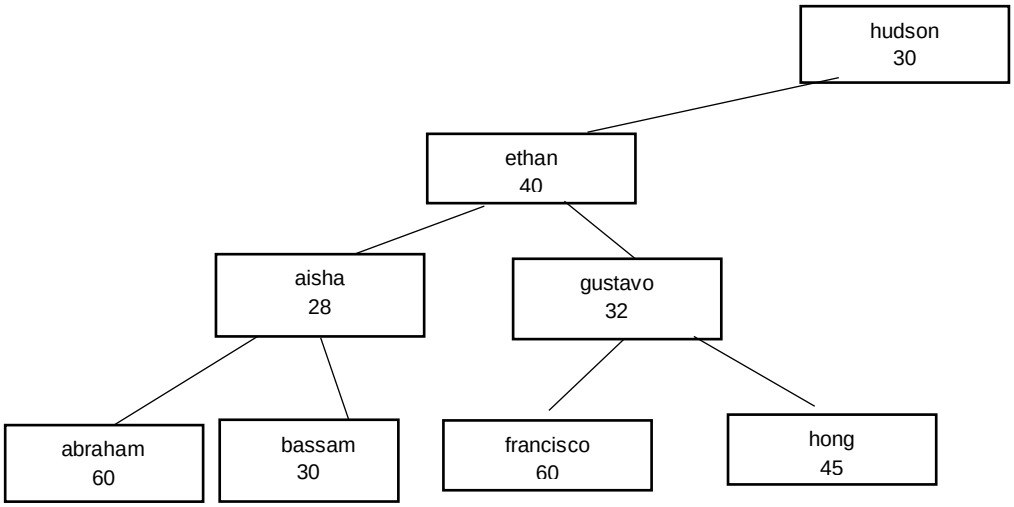
`left height = <lh> right height = <rh> <balance status>`

where `<lh>` is the height of the left subtree, `<rh>` is the height of the right subtree, and `<balance status>` is either “balanced” or “not balanced”.

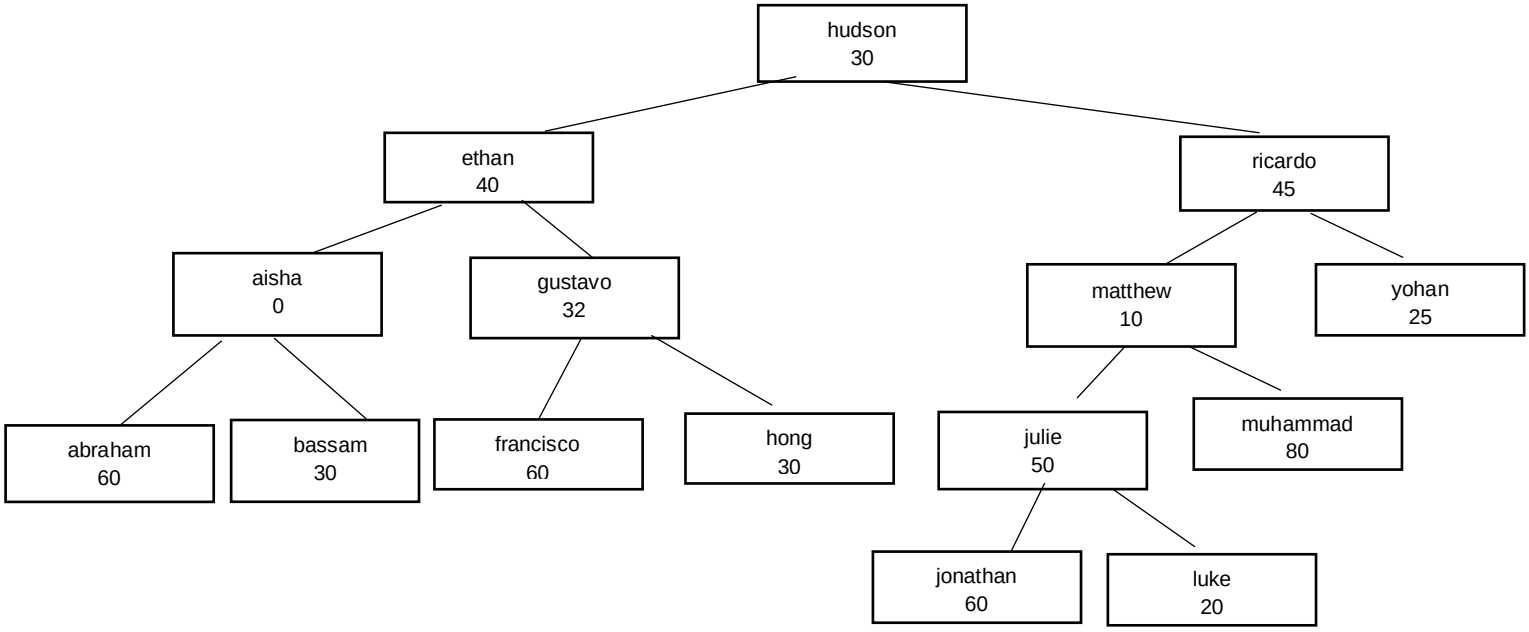
lineNum	Sample Input	Sample Output
1	40	
2	add hudson 30	hudson 30 0
3	add ethan 40	ethan 40 1
4	add gustavo 32	gustavo 32 2
5	add aisha 28	aisha 28 2
6	add fransisco 60	fransisco 60 3
7	add hong 45	hong 45 3
8	add abraham 60	abraham 60 3
9	add bassam 30	bassam 30 3
10	height_balance	left height = 2 right height = -1 not balanced
11	sub aisha 30	aisha 0 2
12	sub hong 15	hong 30 3
13	add gustavo 8	gustavo 40 2
14	add ricardo 20	ricardo 20 1
15	add yohan 25	yohan 25 2
16	add matthew 10	matthew 10 2
17	add julie 50	julie 50 3
18	add muhammad 80	muhammad 80 3
19	height_balance	left height = 2 right height = 2 balanced
20	add jonathan 60	jonathan 60 4
21	height_balance	left height = 2 right height = 3 not balanced
22	add luke 20	luke 20 4
23	del muhammad	muhammad deleted
24	search matthew	matthew 10 2
25	search muhammad	muhammad not found
26	del matthew	matthew deleted
27	height_balance	left height = 2 right height = 2 balanced
28	search julie	julie 50 2
29	del ethan	ethan deleted
30	search bassam	bassam 30 1
31	height_balance	left height = 2 right height = 2 balanced
32	del bassam	bassam deleted
33	height_balance	left height = 2 right height = 2 balanced
34	search aisha	aisha 0 1
35	search fransisco	fransisco 60 3
36	del ethan	ethan not found
37	search gustavo	gustavo 40 2
38	del hudson	hudson deleted
39	search yohan	yohan 25 2
40	search hong	hong 30 0
41	search abraham	abraham 60 2

Sample Explanation

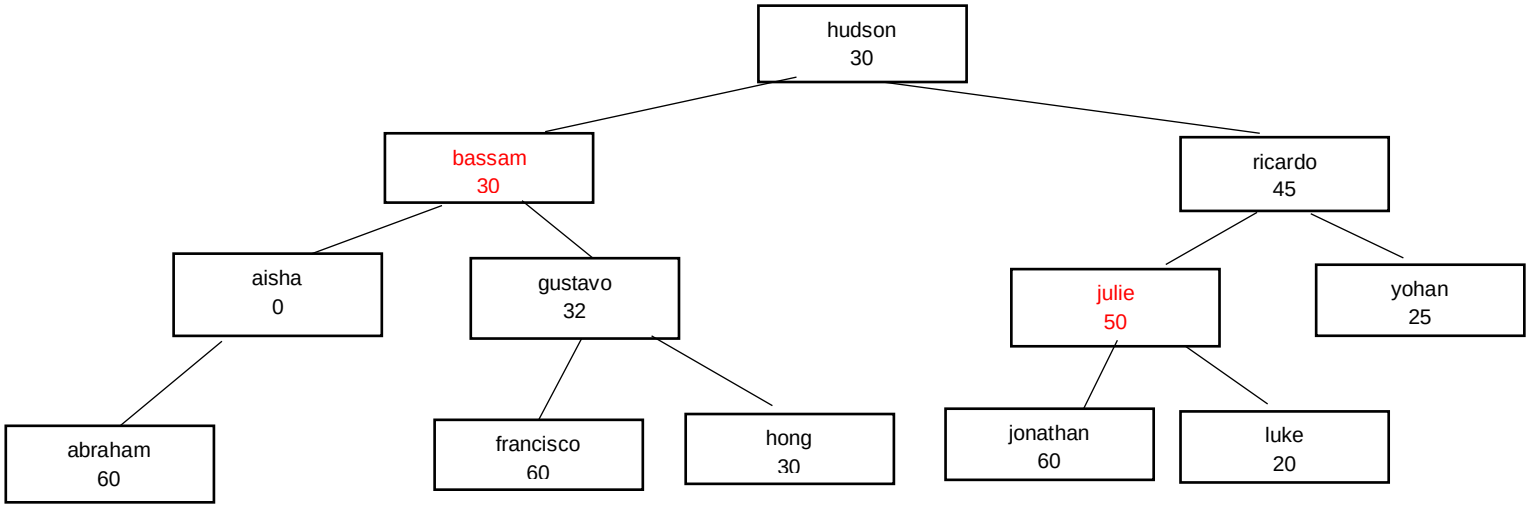
Right before the first sub command (line 11), here is a picture of the tree (without all information stored in each node). The left height is 2 (as the height of the tree starting from ethan is 2) and right height is -1 as there is no right subtree of root of the main tree:



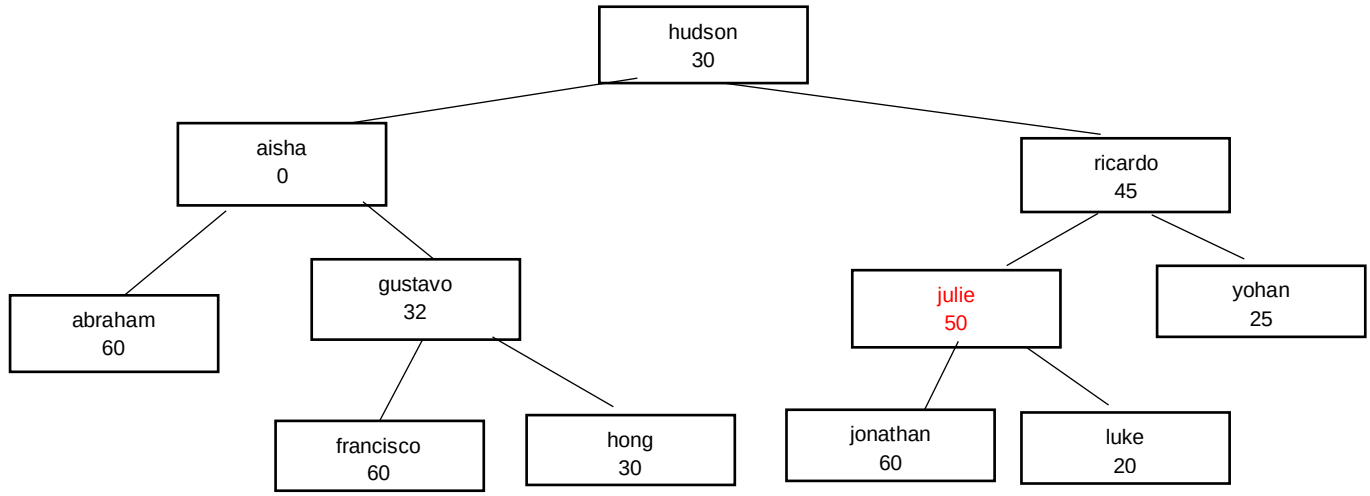
Next, after subtracting all the points from aisha, 15 points from hong, adding 8 points to gustavo, and adding some more customers until the first del command (line 23), our tree becomes like the following. During this time, we have calculated height_balance multiple times to see the status as we keep updating the tree:



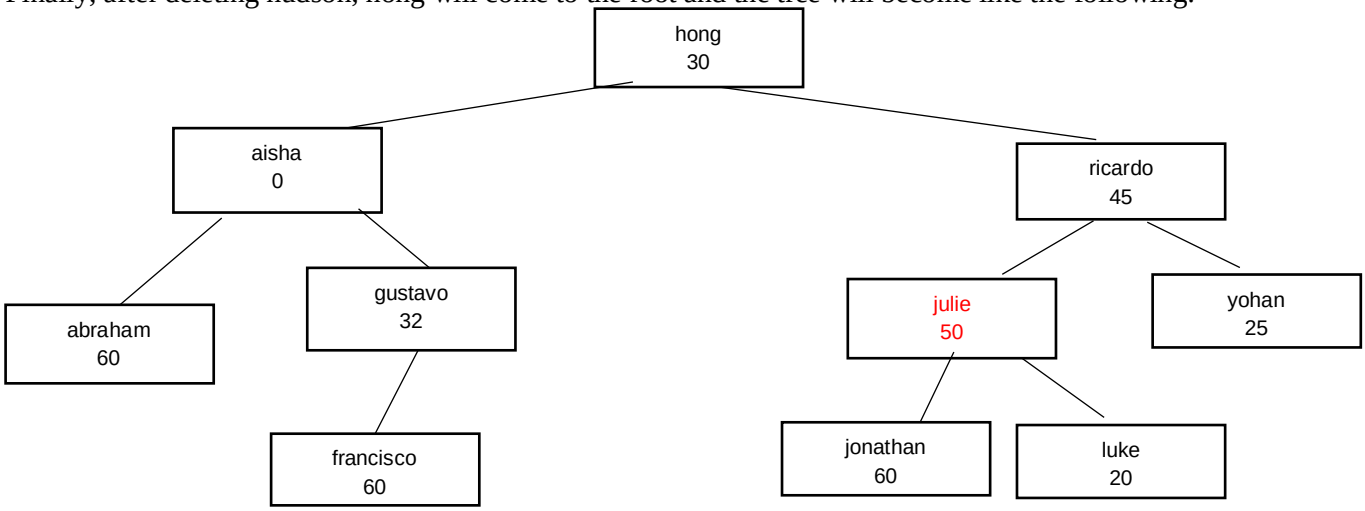
Next, after deleting muhammad (line 23), matthew (line 26), and ethan (line 29), the tree is updated to the following. If you see, after deleing ethan, the place is taken by bassam as the he is the min from the left subtree of etahn. During this time, we have also performed multiple height balance and search operations to see the status of the updated tree. :



Then, on line 32, we have also deleted bassam as well and aisha took the place of bassam as aisha is the max in the left subtree of bassam.



Finally, after deleting hudson, hong will come to the root and the tree will become like the following:



Implementation Requirements/Run Time Requirements

1. Declare a node structure for binary search tree and store all the required inside the node. Do not use any malloc to store the name of a customer. Use static size array for the name with appropriate size based on the max length.

2. The run-time for processing each of the commands should be O(h), where h is the current height of the tree.

3. You must use the delete function we have discussed in our class recording and modify the code to fulfill the requirements of this assignment. It is fine to update the list of parameters and few logic inside the function. However, the logic of dealing with 3 cases should be handled in the same way we have discussed. Not doing this will result in 50% penalty in your assignment.

4. Implement an insert function

5. Implement a search function

6. Implement a Height function

6. You should free all the memory to receive full credit

7. Your code must compile and execute on the Codegrade system.

Few Important Hints:

- Carefully go through the sample input and output and understand the problem clearly based on the explanation
- Use **strcmp function** to compare strings and decide which string comes first alphabetically.
- Work for one command at a time and test it. Obviously, start with search and insertion and after inserting each customer, print them in in-order traversal to check whether they are inserted properly or not
- As you need **search function** during deletion as well as regular search and insertion, a good idea would be returning the node containing the name. Also, you need to calculate depth as you keep searching. A good idea would be using a pointer and pass a reference of a variable initialized with zero and keep updating the value through pointer as you keep traversing the tree while searching.
- As you need to return the root from the **delete function,** you have no option to know whether an item found in the deletion function or not. A good idea also would be using and int pointer in the parameter and pass reference to a flag variable and update the flag to 0 if the name does not exist in the tree or update it to 1 if the name is in the tree. In this way, you will be able to avoid one extra call to the search function before calling the deletion function.
- For height, you can modify the code you have learned in the lab. But, note that the definition of height is a bit different in the lab than regular height definition.

Some Steps to check your output AUTOMATICALLY in a command line in repl.it or other system:

You can run the following commands to check whether your output is exactly matching with the sample output or not.

Step1: Copy the sample output to sample_out.txt file and move it to the server

Step2: compile your code using typical gcc and other commands.

//if you use math.h library, use the -lm option with the gcc command. Also, note that scanf function returns a value depending on the number of inputs. If you do not use the returned value of the scanf, gcc command may show warning to all of the scanf. In that case you can use "-Wno-unused-result" option with the gcc command to ignore those warning. So the command for compiling your code would be:

 # gcc main.c leak_detector_c.c -Wno-unused-result -lm

Step3: Execute your code and pass the sample input file as a input and generate the output into another file with the following command

$ ***./a.out < sample_in.txt > out.txt***

Step4: Run the following command to compare your out.txt file with the sample output file

```
$cmp out.txt sample_out.txt
```

The command will not produce any output if the files contain exactly same data. Otherwise, it will tell you the first mismatched byte with the line number.

Step4(Alternative): Run the following command to compare your out.txt file with the sample output file

```
$diff -y out.txt sample_out.txt
```

The command will not produce any output if the files contain exactly same data. Otherwise, it will tell you the all the mismatches with more details compared to cmp command.

 # diff -c myout1.txt sample_out1.txt //this command will show ! symbol to the unmatched lines.