Parking Violation Tracker

Background Story

The Monster Campus Parking Authority is known for issuing tickets for various parking violations, including reverse parking, parking in restricted areas, and occupying reserved spaces. The campus has greatly benefited from the parking garage systems you developed in the past four assignments. Now, the Parking Authority is looking to build a new system to track parking tickets and fines issued to vehicle owners. The goal is to reduce traffic violations by recording every ticket issued and the associated fines. As you learned in class, storing data in a binary search tree (BST) can improve search efficiency, making this a great opportunity to apply that knowledge.

In the new system, vehicle owners can accumulate fines, pay them off partially, or check their current balance. Additionally, some owners may be removed from the tracking system if they no longer owe fines. The system should also support analytical queries, such as calculating the total and average fines.

In your binary search tree implementation, each node will store the vehicle owner's name and the amount of the fine they need to pay. The owner's name will serve as the key for the BST, with comparisons made based on alphabetical order. As you develop this application, you may also want to implement debugging queries to gain better insights into the structure of the tree.

Problem

Develop a program that processes input commands to manage the parking ticket fine system. The program should respond appropriately to each command, as outlined below:

- *Add a fine to a vehicle owner*
- *Deduct a fine from a vehicle owner*
- *Search for a vehicle owner's record*
- *Calculate the average fine amount per owner in the system*
- *Calculate the total amount of fines in the system*
- *Determine the balance of the tree in terms of height*
- *Calculate the total fine amount for vehicle owners whose names are alphabetically smaller than a given name*

Input (must be standard input using scanf (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. Note that all strings in this assignment are assumed to be single-word strings. Therefore, using scanf with %s should be sufficient to capture each string input. Below is the format for each of the possible input lines:

Command 1

`add <name> <fine>`

- `<name>` A lowercase alphabetic string containing no more than 25 characters.
- `<fine>` A positive integer, less than or equal to 100.

This command adds a vehicle owner with the specified name (`<name>`) to the tree. If the owner already exists in the system, the command will increase the owner's fine by the given amount (`<fine>`).

Command 2

`deduct <name> <fine>`

- `<name>` A lowercase alphabetic string containing no more than 25 characters.
- `<fine>` A positive integer, less than or equal to 100.

Behavior:

- If the vehicle owner has a fine amount less than the specified deduction, the fine will be reduced to zero (or the owner's current fine if it's less than the specified amount).
- If the owner's fine becomes zero or negative after the deduction, the owner is removed from the tree.

This command ensures that if a deduction exceeds the current fine, the fine is set to zero and the owner is removed from the system if they no longer owe any fines.

Command 3

`search <name>`

- `<name>` A lowercase alphabetic string containing no more than 25 characters.

Behavior:

- This command searches for the vehicle owner with the specified name `<name>`) in the tree.
- If the owner is found, the command will report:
  - The remaining fine the owner still needs to pay.
  - The depth (or level) of the node in the tree that stores that owner's record.

The depth refers to the number of edges from the root node to the node storing the vehicle owner's record.

Command 4

`average`

This command calculates the average amount of fine per vehicle owner in the tree.

Command 5

`height_balance`

This command calculates and compares the heights of the left and right subtrees of the root node.

Behavior:

- The command will calculate the height of the left subtree starting from the left child of the root node.
- It will also calculate the height of the right subtree starting from the right child of the root node.
- If the heights of the left and right subtrees are the same, the command will output that the tree is "balanced."
- If the heights are different, the command will output that the tree is "not balanced."

Command 6

`calc_below <name>`

- `<name>` A lowercase alphabetic string containing no more than 25 characters.

Behavior:

- This command calculates the total amount of fines for all vehicle owners whose names come **alphabetically** before the specified <name>.
- The calculation includes only those owners whose names are lexicographically smaller than equal to <name>, based on alphabetical order.
- The command should sum up the fines of all such vehicle owners in the system and output the total amount.

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

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

Commands 1

Print out a single line with the format:

`<name> <fine> <depth>`

The <name> refers to the vehicle owner's name who is being fined, <fine> represents the updated total fine they owe, and <depth> indicates the depth of the node in the tree where the owner's record is stored. If the vehicle owner does not already exist in the tree, the program should insert the owner and then display the message with the owner's name, their new fine, and the depth of the corresponding node in the tree.

Commands 2

Print out a single line with the format:

`<name> <fine> <depth>`

The `<name>` represents the vehicle owner's name whose fine has been deducted, `<fine>` is the updated total fine they now owe, and `<depth>` is the depth of the node where their record is stored. If the deduction causes the owner's total fine to reach zero or become negative, the owner should be removed from the tree, and the following message should be printed:

`<name> removed`

If the vehicle owner does not exist in the tree when the deduction command is executed, the program should print:

`<name> not found`

Important note: If the deduction results in deleting a vehicle owner from the tree, and the owner's node has two children, replace the deleted node with the maximum node from the left subtree. This ensures that there is a consistent and correct result for each test case.

Command 3

If the vehicle owner 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> <fine> <depth>`

The `<name>` refers to the vehicle owner's name being searched for, `<fine>` is the total fine they owe, and `<depth>` indicates the depth of the node where their record is stored in the tree.

Command 4

Show the average up to two decimal places. Please use double data type for all fractional calculations and also make sure not to do int division by mistake.

`<average>`

where `<average>` is the average amount of fine per vehicle owner available in 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".

Command 6

`<total>`

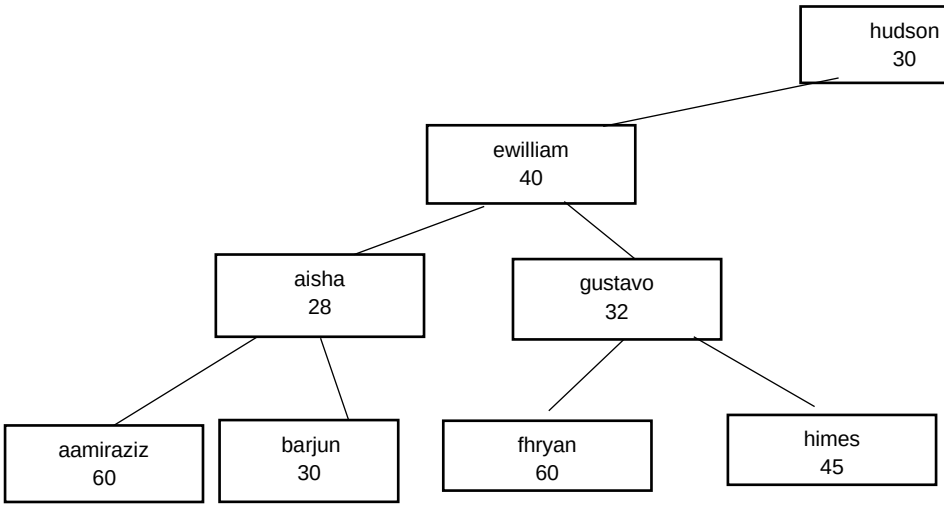
Where `<total>` is the total amount of fines for the vehicle owners whose names are **lexicographically less than or equal to** the name provided in the command. Note that the name entered in the command **does not necessarily have to be present in the tree**. The total should include fines for all owners whose names come before or are the same as the given name in alphabetical order.

Sample input/output on the next page

Line	Sample Input	Sample Output
1	46	
2	add hudson 30	hudson 30 0
3	add ewillam 40	ewillam 40 1
4	add gustavo 32	gustavo 32 2
5	add aisha 28	aisha 28 2
6	add fhryan 60	fhryan 60 3
7	add himes 45	himes 45 3
8	add aamiraziz 60	aamiraziz 60 3
9	add barjun 30	barjun 30 3
10	height_balance	left height = 2 right height = -1 not balanced
11	calc_below happy	250
12	average	40.62
13	deduct aisha 23	aisha 5 2
14	deduct himes 15	himes 30 3
15	add gustavo 8	gustavo 40 2
16	add ricardo 20	ricardo 20 1
17	add zkevin 25	zkevin 25 2
18	add matthew 10	matthew 10 2
19	add janeesha 50	janeesha 50 3
20	add muhammad 80	muhammad 80 3
21	height_balance	left height = 2 right height = 2 balanced
22	add james 60	james 60 4
23	height_balance	left height = 2 right height = 3 not balanced
24	add luke 20	luke 20 4
25	calc_below ricardo	535
26	calc_below muhammad	515
27	average	37.33
28	deduct muhammad 80	muhammad removed
29	search matthew	matthew 10 2
30	search muhammad	muhammad not found
31	deduct matthew 15	matthew removed
32	height_balance	left height = 2 right height = 2 balanced
33	search janeesha	janeesha 50 2
34	deduct ewillam 41	ewillam removed
35	search barjun	barjun 30 1
36	height_balance	left height = 2 right height = 2 balanced
37	deduct barjun 30	barjun removed
38	height_balance	left height = 2 right height = 2 balanced
39	search aisha	aisha 5 1
40	search fhryan	fhryan 60 3
41	deduct ewillam 20	ewillam not found
42	search gustavo	gustavo 40 2
43	deduct hudson 30	hudson removed
44	search zkevin	zkevin 25 2
45	search himes	himes 30 0
46	calc_below himes	195
47	calc_below aamiraziz	60

Sample Explanation

Right before the first deduct command (line 13), 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 ewillam is 2) and right height is -1 as there is no right subtree of root of the main tree. Also, "calc_below of happy" is calculated by summing up all the fines except hudson and himes as they are not alphabetically smaller or equal to "happy":



Next, after deducting 23 from aisha, 15 from himes, adding 8 to gustavo, and adding some more vehicle owners until the deduct command of muhammad (line 28), 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. Also, "*calc_below ricardo*" gave the sum of all the fines except *zkevin* from the following tree. Also, "*calc_below muhammad*" gave the sum all the fines except *ricardo* and *zkevin* :