Corrupt Captors

Objective

Give practice with Binary Search Trees in C
Give practice with Tree Rotations in C

Story

BTS has been rounding up the raccoons KAT pet shop initially set out to apprehend. BTS has been completing the job surprisingly well. You decided to infiltrate BTS headquarters and you found that they are behind the raccoon disaster.

BTS have been placing raccoons in town and capturing them once a certain amount of food has been taken. The food that the raccoons have taken is then sold at BTS for a nice profit. You are going to put an end to this, but first you want to determine based on the data how much food has been acquired dishonestly.

It's not as easy as it sounds. According to what you recovered from BTS, raccoons form a hierarchy based on both their location and their hat sizes; as we all know, raccoons love wearing fancy hats.

In summary, imagine raccoons living along a line, depicted as a number line in this scenario. Each raccoon will be given a distinct specific spot on this line to call home. Additionally, each raccoon wears a hat of a unique size, signifying its importance. The raccoon with the biggest hat in a location is the leader, and all raccoons ultimately report to this leader, either directly or indirectly.

The leader raccoon divides the line into two sections: one with values less than its location and the other with values greater. Raccoons on opposite sides of the leader raccoon don't interact. Instead, the raccoon with the largest hat on each side becomes the leader for its respective section of the line, excluding the current leader. This hierarchy extends downward until reaching raccoons without any subordinates.

BTS to control the raccoon hierarchy would occasionally place new hats near raccoons. The closest raccoon to the spot nearest the hat would take the hat. With equidistant raccoons, the one with the smallest hat arrives first to claim the hat. If the hat is worse than the current hat of the raccoon, they would destroy it.

When food is taken at a location, it's grabbed by the raccoon nearest to that spot on the number line. If two raccoons are equidistant, the one with the smallest hat arrives first to claim the food. Then, half of the food (rounded up) is passed to the raccoon's direct leader, while the remaining half is kept by the raccoon who initially grabbed it.

Subsequently, each raccoon that receives food from its subordinate shares half of it (rounded up) with its direct leader and keeps the rest.

When a raccoon is captured the food it has amassed is also take by BTS, and the hat of the raccoon is either taken and worn/destroyed by the nearest raccoon, or taken by BTS if no raccoon is left.

Problem

You will be given a list of commands that can be one of the following:

- BTS adds a raccoon
- BTS captures a raccoon
- BTS adds a hat
- Raccoon steals food
- Stop program

You need to determine for each capture command how much pet food was recovered from capturing the raccoon. Additionally, after all commands are over print how much food is left over at each location. The locations can be printed in any order.

Input

Following this will be a series of command in 1 of the following 5 formats,

- `ADD` ***X H***
  - Which represents BTS putting a raccoon at location ***X*** with a hat size of ***H***. No raccoon will currently be at location ***X***. No raccoon will currently have a hat size of ***H***.
- `CAPTURE` ***X***
  - Which represents BTS capturing a raccoon at location ***X***. It is guaranteed that a raccoon will be at position ***X***.
- `HAT` ***X H***
  - Which represents BTS placing a hat of size ***H*** at location ***X***. This command will only happen when a raccoon is present. No raccoon will currently have a hat size of ***H***.
- `STEAL` ***X A***
  - Which represents a raccoon stealing ***A*** food from location ***X***. This command will only happen when a raccoon is present.
- `QUIT`
  - Which represents the end of the commands.

For bounds,

- Food amount will be positive integers up to 1,000,000.
- Hat sizes will be positive integers up to 1,000,000,000.
- Locations will be integers of magnitude no more than 1,000,000,000.
- The raccoons are structured, but you can assume that there won't be a raccoon chain of command longer than 200 critters.

Output

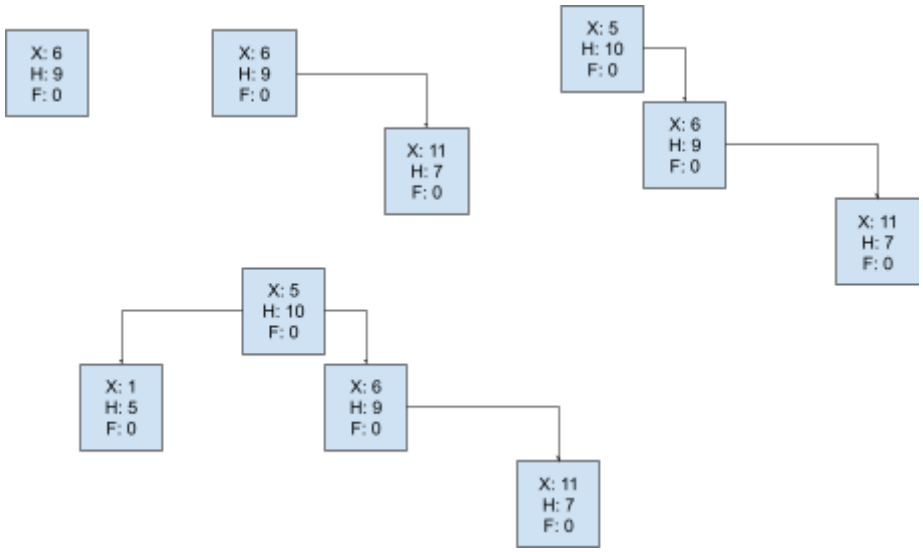
For each capture command print on a line by itself an integer representing the amount of food recovered from the raccoon captured. After the quit command for each remaining raccoon, output its location and the amount of food it has, space separated. The raccoons printed at the end can be in any order.

Sample Input	Sample Output
ADD 6 9 ADD 11 7 ADD 5 10 ADD 1 5 STEAL 13 3 CAPTURE 6 STEAL 3 10 HAT 2 8 CAPTURE 5 STEAL 4 9 QUIT	1 6 1 14 11 1
ADD 5 6 STEAL 10 5 ADD 10 10 STEAL 10 5 ADD 7 27 STEAL 10 5 HAT 6 30 STEAL 10 5 QUIT	5 7 7 4 10 9

Sample Explanation

FOR CASE 1

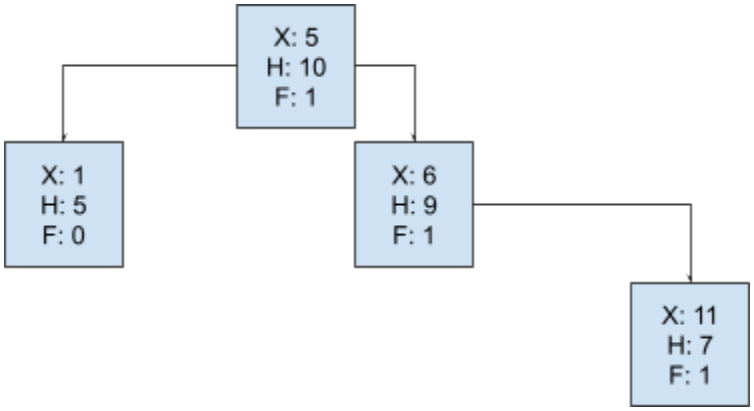
Four raccoons are added before anything else. The raccoon hierarchy goes through the following structures as the additions occur.



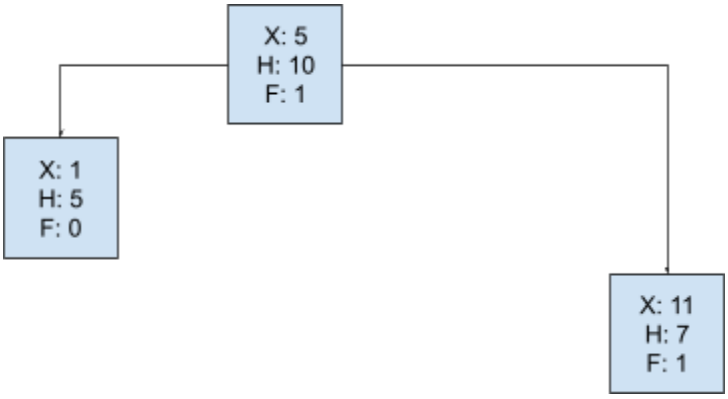
X represents the location of the raccoon; H represents the size of the hat of the raccoon; and F represents the amount of food the raccoon has.

The first command after the additions is 3 food stolen from location 13. The raccoon at location 11 is able to take that food. They keep 1 and pass 2 to their overseer–raccoon at location 6. The raccoon at location 6 keeps 1 of it and passes 1 to their overseer–raccoon at location 5.

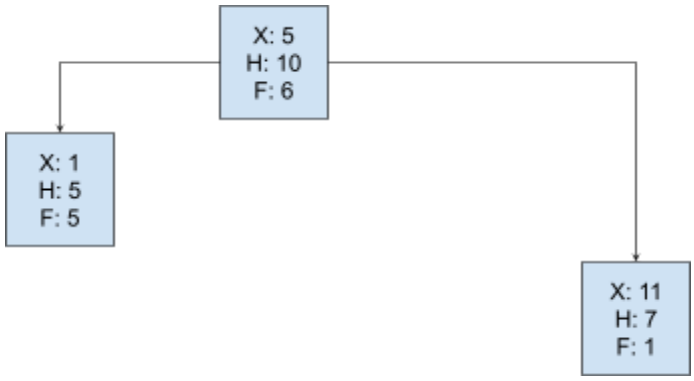
Below is an example of the raccoon hierarchy after the steal command.



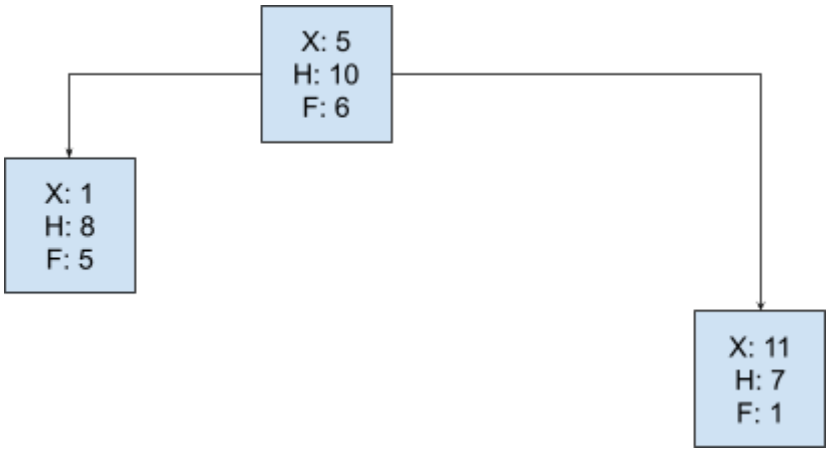
The next command captures the raccoon at location 6. The amount of food recovered is 1. The value of 1 is printed. Raccoon at location 5 steals the remaining hat and destroys it. The structure of the raccoon hierarchy looks like the following after the capture command.



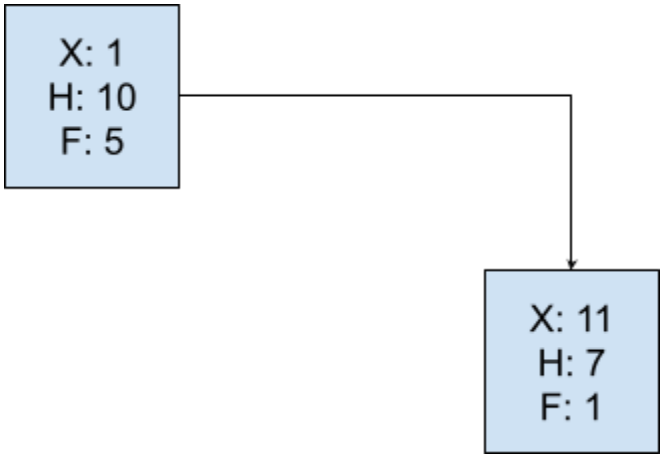
The next command is 10 food stolen from location 3. Raccoon at location 1 gets the food and keeps 5 and passes 5 to raccoon at location 5. The following structure results.



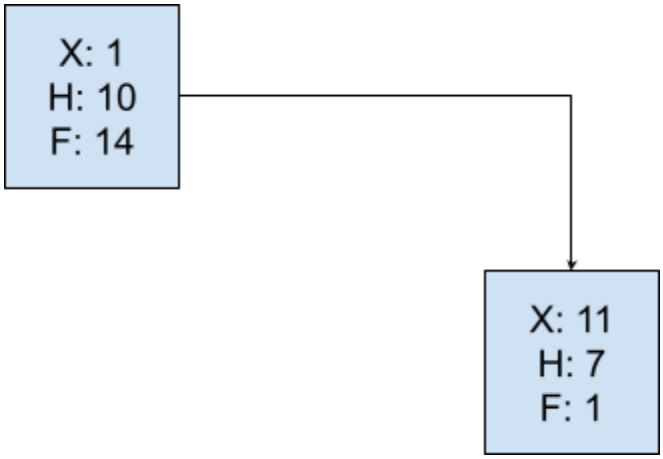
The next command places a hat at location 2 with a size of 8. The raccoon at location 1 steals the hat, but does not change their position in the hierarchy. The following structure is the result of the command.



The next command captures the raccoon at location 5. BTS recovers 6 units of food. Raccoon at location 1 steals the hat left behind and wears it. The result is the following structure.



The next command is 9 units of food stolen from location 4. This is taken and all kept by the raccoon at location 1. The result is the following structure.



FOR CASE 2

The first theft is taken all by the raccoon at location 5. The second theft is taken all by the raccoon at location 10. The third theft has 2 food units kept by the raccoon at location 10 and 3 food units kept by the raccoon at location 7. The fourth theft has 2 food units kept by the raccoon at location 10, 1 food unit kept by the raccoon in location 7, and 2 food units kept by the raccoon at location 5.

At the end (with no captures), the raccoon at location 5 has a total of 7 food units; the raccoon at location 7 has 4 food units; and the raccoon at location 10 has a total of 9 food units.

Hints

Raccoon Tree: The raccoon should be stored in a tree, where the hierarchy reflects the structure. The tree is a BST with the location number as the search value.

Insertion: The add command is a BST insertion with a potential hierarchy change that can be accomplished via tree rotations.

Deletion: The capture command is a BST deletion with a potential hierarchy change that can also be accomplished via tree rotations.

Hat Size Hierarchy: My recommendation is that parent nodes should always have a larger hat size than their children.

Tree Rotations: Tree rotations can be used to keep the same order, but change the hierarchical structure. The hierarchy can change when adding a raccoon, capturing a raccoon, or changing the hat size of a raccoon. When a raccoon has a larger hat than its parent, the parent should rotate away from the larger hatted child raccoon.

Find Closest: Finding the closest raccoon is very important for many of the functions. To find the closest raccoon to a given location on the number line, you can traverse the BST based on the raccoons' locations. As you traverse the tree, compare the current raccoon's location with the target location to determine whether to move left or right in the tree. Keep track of the closest raccoon encountered during the traversal. This approach ensures that you find the raccoon closest to the specified location efficiently. This is easier to do iteratively than recursively.

Command Handling: Consider using a function to handle each of the commands. Including reading in input.

Tree Traversal: To print all the raccoon at the end of the program consider using a traversal to ensure each location is visited.

Node Struct Example: Consider using the following node struct,

```
typedef struct Node Node;
struct Node {
   Node * left, * right, * parent;
   int hat;
   int x;
   long long int food;
};
```

Function Prototype Examples: Consider using the following function prototypes,

```
Node * createNode(int location, int hatSize);
Node * insertRaccoon(Node * root, int location, int hatSize);
Node * captureRaccoon(Node * root, int location);
Node * changeHat(Node * current, int hatSize);
void   stealFood(Node * root, int location, int amt);
Node * findClosest(Node * root, int location);
Node * rotateLeft(Node * current);
Node * rotateRight(Node * current);
void   printTree(Node * root);
```

Grading Criteria

- Read/Write from/to **standard** input/output (e.g. scanf/printf and no FILE *)
    - 5 points
- Good comments, whitespace, and variable names
    - 15 points
- No extra input output (e.g. input prompts, “Please enter the number of cats:”)
    - 5 points
- Use a BST
    - 5 points
- Implement Tree Rotation based on hat sizes
    - 5 points
- Support Node Insertion
    - 5 points
- Support Node Removal
    - 5 points
- Process commands until QUIT
    - 5 points
- Programs will be tested on 10 cases
    - 5 points each

No points will be awarded to programs that do not compile using “gcc -std=gnu11 -lm”.

Sometimes a requested technique will be given, and solutions without the requested technique will have their maximum points total reduced. For this problem implement a BST. ***Without this, programs will earn at most 50 points!***

Any case that causes a program to return a non-zero return code will be treated as wrong. Additionally, any case that takes longer than the maximum allowed time (the max of {5 times my solutions time, 10 seconds}) will also be treated as wrong.

No partial credit will be awarded for an incorrect case.