Computer Science Program: Binary Belle

Everyone knows that Belle, from Beauty and the Beast, is rather peculiar; she has her head in books and she falls in love with a beast. What most people don't know is that how she stores her files on her computer is quite peculiar as well. Belle has real trouble giving her directories names. She has decided that rather than fret about what to name her directories, every single one of them will either be named "L" or "R", essentially like the left or right child in a binary tree!

This means that in a single directory will contain either

(a) one file
(b) two files
(c) one directory
(d) two directories
(e) one directory and one file

She never creates a directory that stays empty and she never places more than two items in any directory! This is precisely how she got the nickname "Binary Belle" when she was studying Computer Science in her undergraduate years. (Even Walt Disney wasn't aware of this little known nickname!)

Just like most users, Belle does the following things on her computer:

(1) creates files/directories
(2) deletes files/directories (directory deletes remove everything in the designated subtree)
(3) queries the number of files in a directory
(4) queries the number of subdirectories in a directory (not counting it)
(5) queries the total size of a file/directory

The name of each directory (except the root directory) will be either "L" or "R" and the size of each directory will be given. (This is just the size to store the directory, not any of its contents.) The root directory is named "C" and its ***size is 17 bytes.***

The name and size of each file will be given. Each name will be a string of 2-19 characters from the following set: letters, digits, underscore, period.

The total size of a directory is the sum of the sizes of the files and directories within it, including itself.

Belle never nests her directories more than 100 levels deep.

Suggested Struct

You must create a binary tree node struct of some sort for this assignment. If you are not sure how to create your struct, please use the following one:

```
typedef struct filenode {
    int isDir;
    char name[MAX+1];
    struct filenode* left;
    struct filenode* right;
    struct filenode* parent;
    int numFiles;
    int numSubDir;
    int totalSize;
} filenode;
```

Notice that the design here is that we store extra information in nodes so that answering queries is easier and traversing down and up the tree is easier also. (Note: Depending on how the insert and delete are coded, the parent link may not be necessary. But, if you don't carefully think about the design of those functions, having the parent node is very helpful and can "save" you.)

The trade-off for having all of this information stored in each single node is (a) more memory use and (b) more information to properly maintain when any change (insert or delete) is made to the tree.

The Problem

Trace through a sequence of changes that Belle makes to her directory structure. For each query in the sequence of instructions, output the requested value on a line by itself. At the beginning of each input case, Belle begins with a root directory. Each instruction will be valid - when an item is created it will be inserted into an existing directory.

Difference in Testing

Unlike the previous assignments, your program will be tested on a single input case, but it will be run multiple times so that we can test different input cases. Thus, in the input format, there will NOT be an integer representing the number of test cases. Your program, when run, should simply process a single input case. When we test your program, we will test it on several different input files (piped into standard input as usual).

Input Format

The first line of input will be the number of total instructions (file/directory creations and deletes and queries). The instructions will follow, one per line. Each instruction will be several space separated tokens.

The syntax for each instruction in the input is as follows:

All create instructions will start with the integer 1. This will be followed by a string designating the location in the directory/file structure this item will be in. This string will be a sequence of 1-100 characters from the set {'L', 'R'}. This string will be followed by the name of the file or directory. If the item is a directory, the name will be the last character of the location string. If the item is a file, it will be the name of the file (in between 2 and 19 characters long, as previously designated). Thus, you can differentiate between a file and a directory by the length of the name. This will be followed by an integer, representing the size of the item in bytes. Though a create may initially make an empty directory, these directories will later be filled with at least one item.

All delete instructions will start with the integer 2. This will be followed by a string designating the location in the directory/file structure this item is at. This string will be a sequence of 1-100 characters from the set {'L', 'R'}. (So we never delete the root directory.) Also, deletes will be such that they never leave empty directories.

All queries for the number of files in a directory will start with the integer 3. This will be followed by a string designating the location of the directory. If the directory is a subdirectory of the root, this string will be a sequence of 1-100 characters from the set {'L', 'R'}. If the query is for the root directory, the string will be "C". You will always be given a string that corresponds to the location of a directory for this type of query.

All queries for the number of subdirectories in a directory will start with the integer 4. This will be followed by a string designating the location of the directory. If the directory is a subdirectory of the root, this string will be a sequence of 1-100 characters from the set {'L', 'R'}. If the query is for the root directory, the string will be "C". You will always be given a string that corresponds to the location of a directory for this type of query. Note that this value could be zero, if the directory queried stores either 1 or 2 files.

All queries for the total size of a file/directory will start with the integer 5. This will be followed by a string designating the location of the file/directory. If the directory is a subdirectory of the root or a file, this string will be a sequence of 1-100 characters from the set {'L', 'R'}. If the query is for the root directory, the string will be "C". You will always be given a string that corresponds a valid item in the directory structure. If the item is a directory, add up the sizes of each item (in bytes) in the subtree of this directory, including the directory itself. If the item is a file, just return the size of that file (in bytes).

Output Format

For each query, out out the result on a line by itself.

Sample Input

```
29
1 L L 17
1 R R 17
1 LL L 17
1 LR Hmk4.doc 12056
1 LLL Numbers.xlsx 33096
1 RR binarybelle.c 8064
3 C
4 C
5 C
3 L
3 R
3 LL
4 L
4 R
4 LL
5 L
5 R
5 LL
5 LR
5 RR
5 LLL
2 L
3 C
4 C
5 C
3 R
4 R
5 R
5 RR
```

Sample Output

```
3
3
53284
2
1
1
1
0
0
45186
8081
33113
12056
8064
33096
1
1
8098
1
0
8081
8064
```

Deliverables

Turn in a single file, *belle.c*, over WebCourses that solves the specified problem. Please do not make any enhancements to your program. Make sure it solves this specified problem exactly.