<u>**Binary Belle - Hints**</u>

All of these hints correspond to using the struct given in the program description.

Here are the critical functions to write, described by task:

1) A function to create a single node of the tree.
2) An insert function, to add a file/directory to the tree structure.
3) A delete function, to delete a file/directory to the tree structure.
4) A function that frees all the nodes associated with a tree pointed to by a filenode pointer.
5) A function the returns the number of files in a directory.
6) A function that returns the number of subdirectories in a directory.
7) A function that returns the size of a file/directory.

Here are some notes about one possible way to design each function (there are other ways to design these functions.):

1) Have this function take in all the necessary information to store a file or directory. The function should dynamically allocate a node and fill it out assuming that this file/directory is the only thing that exists, and then return a pointer to the allocated node.

2) A recommended method of writing this function is doing it recursively. In the recursion, the very first step is to update this subdirectory's # of files, # of subdirectories and total size based on the single file/directory being inserted into it. We do this first because as we go down the path recursively, we know that this file will be in this subdirectory. Then, check to see if we're in the last directory before we put this one in. If we are, then go ahead and do the insert, which means patching the inserted node as the appropriate child of the current node. If we aren't, then recursively insert in the appropriate direction.

3) A recommended method of writing this function is doing it iteratively. First, iterate down to the deleted file/directory. Now that we know its # of files, # of subdirectories and its size, we must subtract these out of each ancestor node. Iterate back to the root of the tree, subracting these three values out of each ancestor node till we reach the root. Then, sever the link between the node to be deleted and its parent. Finally, free all the memory (in a different recursive function) associated with the deleted node.

4) This function works precisely like the binary tree example that frees all the memory for a binary tree in the class example.

5, 6, 7) All these functions are nearly identical - recursively follow the appropriate path down the tree. When you arrive at the appropriate node (base case), return the appropriate value.

<u>Function Prototype Hints</u>
Both the insert and delete functions can be void because before the processing of commands begins, you create a tree with one node (the root directory). Based on the specification, no changes will ever be made to this node, thus, there's no need for either the insert or delete functions to return the "new root " of the resulting tree, since the resulting tree will always have the same exact root node.