In this assignment, you will be parsing a set of commands to build a filesystem tree, and then using information about the file sizes to determine which directory to remove to free up enough disk space to store data like, for example, the latest Large Language Model (LLM).

To obtain this information, you have a transcript of a session exploring the contents of the filesystem. All commands are prefixed with `$`; any output from a command has no such prefix. We use a subset of standard Linux commands to navigate the filesystem:

- `ls -1sp`: List the items–one per line–in the current directory with their size, **appending** a forward slash (`/`) if the item is itself a directory. The size is displayed **first**, following by the item's name. All directories will have size `0`, but a file may also have that size so **do not** depend on that telling you if the item is a directory. You do not need to handle any other variants of `ls`.
- `cd <dirname>`: Changes the current directory to the specified `<dirname>`. `cd ..` goes up to the **parent** directory. `cd /` goes to the root of the filesystem, and we will always start with this command. You **do not** need to handle compound path names (e.g. `cd a/e`).

Each file may have an extension, which is the suffix (end of the string) starting with a `.` (e.g. `.txt`, `.mp4`).

You are given a listing that looks like:

```
$ cd /
$ ls -1sp
0 a/
4561 b.txt
67892 c.mp4
$ cd a
$ ls -1sp
37891 d.jpg
0 e/
0 f/
578932 g.jpg
$ cd e
$ ls -1sp
$ cd ..
$ cd f
$ ls -1sp
27894 h.docx
7899 i.xlsx
```

Write a function `readInput` that takes an `istream` containing the transcript, builds a **tree** of directories and files, and returns that object (e.g. the root node). We strongly **recommend** creating a Node-style class to store each item's information and links to children. For the above example, your tree can be printed as

```
()
|
+-(a)
| |
| +-(d.jpg)
| |
| +-(e)
| |
| +-(f)
| | |
| | +-(h.docx)
| | |
| | +-(i.xlsx)
| |
| +-(g.jpg)
|
+-(b.txt)
|
+-(c.mp4)
```

Depending on how you created are storing the tree, you should be able to use something like the following to print your tree.

```cpp
void printNode(Node * node, int lvl=0) {
    int i;
    for (i = 0; i < lvl; i++)
        cout << "| ";
    cout << endl;
    for (i = 0; i < lvl-1; i++)
        cout << "| ";
    if (lvl > 0)
        cout << "+-";
    cout << "(" << node->name << ")" << endl;
    for (auto node: node->children)
        printNode(node, lvl+1);
```

where `node->name` and `node->children` would be replaced with your method of accessing the name of the node and the container of child node pointers. Then, `printTree(is)` should call `readInput` followed by `printNode`.

Next, create a function `computeSize` that given a path, computes the total size of all files and directories. This is recursive, including the sizes of all files in subdirectories. Recall that the size of an empty directory is zero (0). You will be passed a path to a directory. For example, `computeSize(is, '/') = 725069` and `computeSize(is, '/a/f/') = 35793`.

Create a function `smallestDir` that returns the **full path** of the directory that is the **smallest** directory whose size is at least a target amount (the space we need). Thus, if the target size is 1100 and `x` has size 1200, `y` has size 1000, and `z` has size 1300, we would pick `x` since it is the smallest of those directories ≥ 1100 (`x` and `z`). Then, `smallestDir(is, 35000)` returns `"/a/f"` and `smallestDir(is, 650000)` returns `"/a/"`.

Suppose we wish to only delete videos to free up space. Modify the `smallestDir` function to take a file extension and consider only files with that extension to determine the target directory. You may wish to also update `computeSize` to take an extension. Then, `smallestDir(is, 616000,".jpg")` returns `"/a/"`.

Finally, to free up space, our delete operation will also only remove files from that directories (and its subdirectories) with that extension. If all files in a directory are deleted, that directory will also be deleted. Otherwise, it will not be. This means that an empty subdirectory in the located directory will also be deleted. Write a function to print the files and directories that will be deleted and their size. For example, `listDeleted(is, 650000)` prints:

```
/a/d.jpg
/a/e/
/a/f/h.docx
/a/f/i.xlsx
/a/f/
/a/g.jpg
/a/
```

while `listDeleted(is, 616000,".jpg")` prints:

```
/a/d.jpg
/a/e/
/a/g.jpg
```