Assignment 4

Summary

This assignment is an application of binary trees to the area of bioinformatics. As such a bit of background material is useful, although not necessary. One of the problems that scientists often have to solve is the reconstruction of hierarchical relationships among a group of entities of various kinds, when those relationships are unknown but can be inferred from data.

Some areas in which this type of processing of data has been done include social network analysis and medical imaging. For example, in social network analysis, data is used to try to infer the leadership hierarchy of terrorist cells. Another example is the construction of what people sometimes call the evolutionary tree, the tree that shows which organisms descended from which other organisms over time. A wide range of methods have been used to infer these relationships, and in the modern world, with our ability to sequence the genomes of many animals, genes have been used to discover these evolutionary relationships. In this assignment, we will focus on the evolutionary tree problem.

Suppose that we are given a set of species of organisms, and with each species, there is some numeric value that is uniquely associated with that species. The value could be something related to its genome for example. It really does not matter for our purposes how it is derived. What matters is that this value is somehow indicative of how closely the organism is related to other organisms. Thus, as the input data, we are given a set of pairs consisting of the name of an organism and a score. The data might look like

```
ape 11.0
tiger 50.0
lion  55.0
human 9.0
elephant 76.0
monkey  14.0
```

for example. Of course the names of the species would be more precise and there would be lots of data. But for us, these simple names are sufficient to illustrate the problem. The objective of the assignment is to write a program that will read data in this format and construct a binary tree that represents the relationships among the species, and then print out the tree in a particular unambiguous format. The details follow.

Detailed Requirements

Input File Format

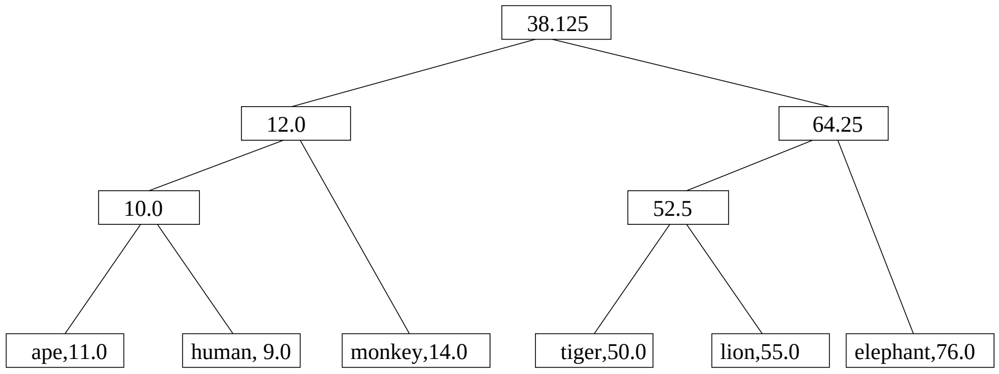
The input data will be in a file whose name is given on the program's command line. The file will consist of some unknown number of lines, each of which contains a string representing the name of an organism, followed by some amount of white space, followed by a decimal number representing its score in some unknown evolutionary weighting scheme. We will call the string the organism's *name* and the numeric value, its *score*.

Processing

The program must construct a binary tree that represents the hierarchical relationship among the organisms. It proceeds as follows. Initially, each (name, score) pair is represented as a tree consisting of a single node. Thus, there will be as many trees as lines in the file.

It then proceeds in iterative stages. At each stage it finds the two trees whose scores are closest to each other and it makes those two trees the left and right subtrees of a new tree. For example, with the data above, human and ape have the closest scores, so it would make a tree with a root and two leaf nodes, one containing the human and the other, the ape. It does not matter which is left and which is right. The score of the resulting tree is the average of the scores of its two subtrees, this case (9.0+11.0)/2, or 10.0. In subsequent iterations, this tree is processed the same way that the rest of the trees: the program searches among all trees for the two whose scores are closest. In our example, it turns out that the monkey and the new tree representing apes and humans have the closest scores, with 14.0 and 10.0 respectively, so it creates a new root node and makes the monkey tree one subtree, and the human/ape tree the other subtree. It sets the score for the root of this tree equal to the average of 10.0 and 14.0, which is 12.0.

This process is repeated until all trees have become part of a single binary tree. Since each iteration reduces the number of trees by one, it eventually must create a single tree. If there were N leaf nodes to start, then after N-1 iterations, all of the original organisms will be leaf nodes in a single tree. The above data would form the tree in the figure below:



Output Format

Once the tree has been constructed, the program will write a representation of it as a string in the language of balanced parentheses. The output will be written to standard output (i.e., `cout` in C++). A binary tree like the one in the figure above, in which every node has either two children or no children, can be represented unambiguously using nothing but left and right parentheses, the strings in the leaf nodes, and commas as separators. We can describe this string recursively as a function of the height of the tree:

1. A tree of height 1 consisting of a root and a left node containing the string '`a`' and a right node containing the string '`b`' is represented by the string "`(a,b)`".
2. If T_1 and T_2 are non-empty trees whose string representations are s_1 and s_2 respectively, then the tree T which has T_1 as its left subtree and T_2 as its right subtree has the string representation "(s_1,s_2)".

Using this recursive definition, the above tree would be represented by the string

```
(((ape,human),monkey),((tiger,lion),elephant))
```

This leads naturally to a recursive function for creating such a string from a binary tree, and it is your job to design and implement that function.

Note. ***The program must not display anything else as output and must not produce any other characters. If it does, this will be considered incorrect output.***

Error Handling Requirements

The program should report if the file named on the command line does not exist or if it cannot be opened for reading. If it finds a line that is not in the correct form (a string followed by a positive number), it should skip it.

Design Considerations

The main program should read and parse the text file and write the output. The simplest approach is to create a list of one-node binary trees as the input is read. Once the input has been read, a separate function should apply the above algorithm to create a single binary tree from the list of these one-node trees. You must create a separate binary tree class for processing the tree, and it should have a function that produces the string representation of the binary tree. That string should be sent to output by the main program. The binary tree class interface and implementation should be in separate files.

Submitting the Assignment

Once your program is finished, you are to create a directory named `hwk4_`*`username`*, where *`username`* is your username on the network. Put all source code files into that directory. Do not put executables, data files, or test files in it. If I find any you will lose three points for each file that does not belong there. Zip up this directory using the UNIX zip command, i.e.,

```
zip -r hwk4_username.zip hwk4_username
```

and put this zip file into `/data/biocs/b/student.accounts/cs235_sw/projects/project4` with octal permission string 600.

You must include either in your main program or in a separate `README` file instructions on how to compile your program, i.e., which files it uses, and so on. Finally, remember to make sure your program meets the requirements of the Programming Rules document.

Before you submit the assignment, make sure that it compiles and runs correctly on one of the `cslab` machines. Do not enhance your program beyond this specification. Do not make it do anything except what is written above.

If you put a file there and then decide to change it before the deadline, just replace it by the new version. Once the deadline has passed, you cannot do this. I will grade whatever version is there at the end of the day on the due date. You cannot resubmit the program after the due date, which is Thursday, December 12.