CS/SE/TE/CE 1337 – Extra Credit Project 2 – The Animal Guessing Game – PART 2

Dr. Doug DeGroot's C++ Classes

Due: April 14, 2021, by midnight. Upload your .cpp file and a PDF copy of your program's saved output file. Because this challenge is for extra credit, please don't ask for any extension to the due date.

Maximum Number of Extra Credit Points:

1. No points if you just implement the file saving function since I've given you the code below; but you might want to add this to your program at some point, even if much later.
2. 10 points if you do both the saving and reading tasks (added to your total score of all tests and homeworks)

Objective:

As we discussed in class, there are times when you want your animal guessing game database to be saved so that you can reuse it and/or correct it. This assignment will focus on

1) saving your database tree and

2) loading/reloading a database into your game.

Those are the two new aspects to your game that you should develop for this project. I am providing some example code below, so there are no extra points provided for simply implementing the file saving portion of this project. Sorry… ☹ However, I will allow extra credit for correctly implementing the file *loading/reading* function and testing your program on a database of your own creation. The database must have at least 10 animals in it and meaningful questions (not silly ones). You can also choose to change subjects from animals to something else meaningful if you'd like. Try books, for example, to see what sort of questions you'd ask.

Discussion:

There are two new features we might want to add to our AnimalGame program:

1. Saving the game's current database (shown below)
2. Loading a specific database before the game begins. This could be our own saved database, a database sent to us by one of our friends, or even one we created by hand.

We will address both these concerns here.

Saving the Database:

To save the tree, we will need two functions:

1. Prepare an output file to print the tree to.
2. Recursively traverse and print the tree using preorder traversal (node, left subtree, right subtree), printing each node as we come to it. We will start with the root node.

There is a discussion of tree traversal and displaying a tree using preorder traversal in our text starting at the end of page 1265. The code there is fine, but it can print only to the screen (console). This is *not* what we want – we want the ability to print the tree to a file so we can

save it and reuse the file later. But come to think of it, we might like to see the tree printed to the console as well as printed to a file. We could write two separate functions to do this, say, something like this:

void displayPreOrder (animalNode* curNode) – using cout, as shown in our text

and

void printPreOrder(animalNode* curNode) – using a file stream that we create

But they'd both do the exact same thing except use different I/O streams. Can we create a single function that prints to *either* the console or to a file? The answer is "yes."

First, let's define a global constant that contains the name of the output file we want to save our database to: something like this:

```
string animalFileName_SAVE = " myAnimalTreeDB.txt ";
```

Now we need to open an output file stream (an ofstream) using that file name:

```
ofstream outfile (animalFileName_SAVE);
```

Now we have an output stream called outfile that we can print to; the output will go to the file called *myAnimalTreeDB.txt* in this example. Notice that even though we created outfile as an *ofstream* (output file stream), the function printTheTree (see below) expects an *ostream* rather than an *ofstream*. That's OK, because an ofstream actually *is* an ostream. And so is cout! We'll see below why that's important.

Here's the function to print the tree. We'll be able to use it to both print to a file and to display the tree on the screen (unlike the code in our text). That's good news, as now we need only one function to see the tree on our screen and/or to save it to a file.

(Note: in the code that follows, I have changed a few things as my code is different from what I suggested you use in your homework. As a result, I may have introduced a bug or three. I hope I haven't, though.)

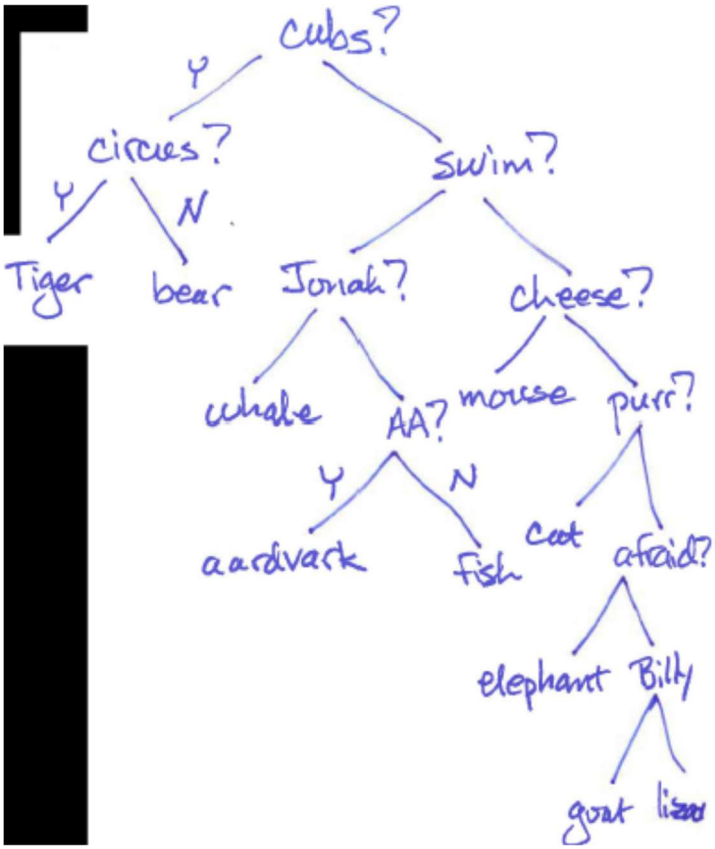
PrintTheTree

```
// printTheTree - print the tree to the output stream in pre-order manner
void printTheTree (ostream &outputStream, animalNode *currentNode) {
    //see if this node is a guess node
    if ((currentNode->yesAns) == nullptr) { // it is
        outputStream << "G" << endl;
        outputStream << currentNode->guess << endl;
    } else { // no, it's a question node; print it and then traverse the children
        outputStream << "Q" << endl;
        outputStream << currentNode->question << endl;
        printTheTree(outputStream, currentNode->yesAns);
        printTheTree(outputStream, currentNode->noAns);
    }
```

} //printTheTree

This will give you an output formatted like this example. This is what the output file will contain.

Q
does it have cubs for babies?
Q
does it perform in the circus?
G
Tiger
G
bear
Q
does it swim?
Q
did it swallow Jonah?
G
whale
G
fish
Q
does it eat cheese?
G
mouse
Q
does it purr?
G
cat
Q
is it afraid of mice?
G
elephant
Q
is it often nicknamed Billy?
G
goat
G
Lizard



Note that the letters Q and G are on lines by themselves. The letter Q precedes a question line, while the letter G precedes a guess line. This is an overly simplistic file format, but hopefully it will make your coding simpler. Because we print the tree using pre-order traversal, we will see a node, then all that node's left subtree, and then all that node's right subtree. This can take some getting used to when reading a pre-order tree output. Below, I have indented the lines to show you the implicit organization of the output, but you don't need to do this in your output file. (I hope have done this correctly.)

```
Q
does it have cubs for babies?
    Q
    does it perform in the circus?
    G
    Tiger
    G
    bear
Q
does it swim?
    Q
    did it swallow Jonah?
        G
        whale
        G
        fish
    Q
    does it eat cheese?
        G
        mouse
        Q
        does it purr?
            G
            cat
            Q
            is it afraid of mice?
                G
                elephant
                Q
                is it often nicknamed Billy?
                    G
                    goat
                    G
                    Lizard
```

Question: Can you figure out a way to use recursion in your in-order tree printing routine to add some initial spaces to each line based on the depth of a node and its data, as shown above? It's easy!

Initiating the File Save

Here's how we would initiate the file-writing process.

```
void saveTheAnimalTree() {
        ofstream outfile (animalFileName_SAVE);
        //check for file-opening errors here
        printTheTree (outfile,rootNode);
        outfile << flush;
} //saveTheAnimalTree
```

Hmm, should I also close the file here?

And Printing the Tree to the Console:

Now let's go back to the problem of printing the tree to the console. We just created a function that prints the tree to an output file. Can we use that same function to print the tree to the console? Yes, we can. Here's the code to do exactly that.

```
printTheTree(cout,rootNode);
```

Because cout is an ostream, we can pass cout as a parameter to our printTheTree function, and we will see the tree printed to the console. Nice!

Reading the File:

Now we can turn our attention to reading a database file. Here I will assume we are reading in the same file as we printed above. The file could be simple and contain only a root node set to lizard, for example. In such a case, the file will look like this:

```
G
Lizard
```

And that's all! Here's another simple, possible input file (with added notation, but no indentation):

```
Q
Does it bark?
G
Dog
Q
Does it purr?
G
Cat
G
lizard
```

which can be read and interpreted as follows:

Q
Does it bark?
[if yes, go down the YES branch to reach a guess node for dog]
G
Dog
[if no, go down the NO branch to read this question node]
Q
Does it purr?
[if yes, go down the YES branch to reach a guess node for cat]
G
Cat
[if no, go down the NO branch to read a guess node for lizard]
G
lizard

Reading a Saved Database:

To read a tree database, we simply perform the opposite of saving it. I'll leave this task up to you. But it's relatively simple. As you read the saved tree file, you will read either a Q or a G line, and then either a question or a guess, respectively. As you read guesses and questions, you reconstruct the tree – again, in pre-order.

How to Submit:

If you implement both the save and read functions, you can submit this project for extra credit. Create a zip file that contains

1. your entire program (the .cpp file)
2. any and all tree database files you used to test your program
3. and a text file containing a copy of the output of at least one execution of your program after you have read one of the input files and added some new data to it.

Upload the file to eLearning site under the ExtraCredit2 assignment. Remember, this project is optional, so if you don't do it, there is nothing you need to do. However, if you *do* do this, name your file as follows:

EC2-CS1337-Suzy-Smith.cpp

and

EC2-CS1337-Suzy-Smith.pdf (for your saved database file)