Name: Jacob Isbell
Date: 4/25/2012
Assignment: Program 4 - CSC2110 - Dr. Eberle

Pseudo-code of Algorithm:

```
main.cpp

    Welcome message
    Create/call Tree class
    Display nice exit message

BST.h

    Define treeInsert, treeSave and playGame
    Default constructors

    TreeInsert -
        Read if 1 or 0 on line, set isQuestion accordingly, de-increment line length
        Set info to rest of string on the line
        if (node is question) means not to end of tree so call treeInsert again

    TreeSave -
        Gets passed node and streamOutput
        Saves bool first, then info, the ends the line
        Same as insert, if isQuestion, call TreeSave again

    playGame -
        While loop for playAgain
        Open database file or throw error if not found/can't open
        While loop to keep asking user question if it is a question

        if the user says that is not the animal, save response into the BST
        Ignore the \n!!!!


BSTNode.h

    Has several data members -
        info - string
        isQuestion - bool
        left and right pointers of type NodeType
```

Summary:

Design Decisions

One of my first design decisions was to keep the program simple and clean by using a minimal number of files and references. I ended up requiring a total of four files: a main.cpp file, a BST.h file, a BSTNode.h file and a database.dat file.

The hardest design decisions were in regard to how to read and write the binary search tree to a file. I decided on a very simple method for storing a tree. First, I made the extension on the database file .dat so that users will not be able to freely open it and make harmful changes to it as easily as a plain text (.txt) file. Another decision I made was in regard to how I know whether or not the node of a tree is a question or an answer. My solution was to create a boolean variable in the class of the binary search tree node. This variable will tell whether or not the string contained is a question or an answer. When the tree is written to the database file it first writes a 1 or 0 to the line to mark this property. It then proceeds to write the string to the same line. I thought of other ways of doing this such as detecting if the string has a space (to see if it is a multi-word sentence or just a single word animal) but this method is far easier and much better from a bullet-proof, no buggy-ness program.

Issues

I really didn't run into many issues but one that drove me nuts for about an hour was the classic hidden return characters in the program. The program kept skipping about 8 of the cin statements and just filled them in automatically. It took me a while to figure it out, but I finally figured out it was the hidden return characters.

Something I didn't foresee in my original algorithm was where I need to call the save tree function. Initially, I had it outside the outside of the main game's logic while loop so if the user said yes, they want to play again, the tree would never get saved. Quick fix, just put the three lines of the save function call up into the while loop.

Notes

I decided not to have separate .cpp files for the two classes because it just adds complexity and I didn't feel as if the functions were big enough to justify a completely different file for the functions. I also included the verbal requirements of a giraffe and monkey. I have designed the program to be delivered to the user with the default database file already populated.

I have re-worked the code of this program several times and this is the cleanest, most efficient way I can think of to code this program given the required specifications. I am very proud of my beautiful, clean code!

Bonus video link: http://www.youtube.com/watch?v=9MQXnavTx8w