```c
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include "util.h"
#include "animals.h"

//Play one round of the game (involves recursive traversal to leaf)
void PlayRound(Node * root)
{
    //Play the node we’re at.  Is it a question or guess node?
    if(root->yes && root->no) //Not a leaf, so a question
        if (GetYorN(root->text))
            PlayRound(root->yes);
        else
            PlayRound(root->no);
    else //At leaf.  Need to play leaf
        PlayLeaf(root);
}

//Handle a leaf node (guess)
void PlayLeaf(Node * leaf)
{
    char szBuff[INPUTBUFFSIZE] = ""; //Input buffer
    char szName[INPUTBUFFSIZE] = ""; //Input buffer for animal name
    char szQuestion[INPUTBUFFSIZE] = ""; //Input buffer for animal question

    //We’re at a leaf, which means guess
    sprintf(szBuff,"Is it a(n) %s?",leaf->text);
    if(GetYorN (szBuff))
    {
        //Succesful guess!
        printf("Yay!\n");
    }
    else
    {
        GetString("Oh, no. What was it? ", szName, INPUTBUFFSIZE);
        sprintf(szBuff,
                "What is a yes or no question that would distinguish between \n%s and %s?",
                leaf->text, szName);
        GetString(szBuff,szQuestion,INPUTBUFFSIZE);
        //Create two new nodes
        leaf->yes = (Node *) malloc(sizeof(Node));
        leaf->no = (Node *) malloc(sizeof(Node));
        //Check that we got our memory
        if(!leaf->yes || !leaf->no)
        {
            fprintf(stderr, "Error: DMA Failure.\n");
#ifdef _DEBUG
            fflush(stdin); //Make sure there’s nothing lurking in the buffer.
            printf("Press Enter to Exit");
            fgetc(stdin);
#endif
            exit(EXIT_FAILURE);

        }
        //Ensure leaves are terminated
        leaf->yes->yes = leaf->yes->no = leaf->no->yes = leaf->no->no = NULL;
        sprintf(szBuff,"What would the answer be for a(n) %s?",leaf->text);
        if(GetYorN(szBuff))
            {
                strncpy(leaf->yes->text,leaf->text, ANITEXTSIZE);
                strncpy(leaf->no->text, szName, ANITEXTSIZE);
            }
            else
            {
                strncpy(leaf->no->text, leaf->text, ANITEXTSIZE);
                strncpy(leaf->yes->text,szName, ANITEXTSIZE);
            }
            //Now replace the old leaf with a new question
        strncpy(leaf->text,CleanStringCR(szQuestion), ANITEXTSIZE);
            printf("Noted.  Thanks.\n");
    }
    return;
}

//Display the contents of a tree
void PrintTree(Node * root, int indent)
{
    //We’ll do a fairly simple preorder traversal to print
    //the tree contents, with indenting to represent depth.

    //Ignore NULL nodes
    if(!root) return;

    //Print this node
    printf("%s\n",root->text);

    if(root->yes)
    {
        //Indent, print yes branch
        for(int i = 0; i < indent; ++i)
            printf("-");
        printf("-y->");
        PrintTree(root->yes,indent + PR
INTINDENTSIZE);
    }

    if(root->no)
    {
        //Indent, print no branch
        for(int i = 0; i < indent; ++i)
            printf("-");
        printf("-n->");
        PrintTree(root->no, indent + PRINTINDENTSIZE);
    }
}

//Release memory
Node * Delete(Node * tree)
{
    //Again, we can recursively traverse our tree to
    //release the memory.
    if(tree->yes)
        tree->yes = Delete(tree->yes);
    if(tree->no)
        tree->no = Delete(tree->no);
    free(tree);
    return NULL;
}

//Open a file and call SaveTree to
//save tree to it. DESTROYS EXISTING FILE
void Save(Node * tree, char * filename)
{
    FILE * fp = fopen(filename,"w");
    if(!fp)
    {
        fprintf(stderr, "Error: Unable to open file '%s' for writing.\n",
                filename);
#ifdef _DEBUG
        fflush(stdin); //Make sure there's nothing lurking in the buffer.
        printf("Press Enter to Exit");
        fgetc(stdin);
#endif
        exit(EXIT_FAILURE);

    }
    SaveTree(tree,fp);
    fclose(fp);
}

//Recursively save tree to text file
void SaveTree(Node * tree, FILE * fp)
{
    //If NULL, we're at a leaf, write NULL symbol and return
    if(!tree)
    {
        fprintf(fp,"%s\n",NULLSTRING);
        return;
    }
    //Otherwise, print node text value and recurse
    fprintf(fp,"%s\n",tree->text);
    SaveTree(tree->yes,fp);
    SaveTree(tree->no,fp);
}

//Open a file and call loadtree to
//retrieve a tree
Node * Load(char * filename)
{
    Node * tree = NULL;
    FILE * fp = fopen(filename,"r");
    if(!fp)
    {
        fprintf(stderr, "Error: Unable to open file '%s' for reading.\n",
                filename);
#ifdef _DEBUG
        fflush(stdin); //Make sure there's nothing lurking in the buffer.
        printf("Press Enter to Exit");
        fgetc(stdin);
#endif
        exit(EXIT_FAILURE);
    }
    tree = LoadTree(fp);
    fclose(fp);
    return tree;
}

//Recursively Load tree from a text file
Node * LoadTree(FILE * fp)
{
    //Every line defines a node, in preorder
    char line[ANITEXTSIZE] = "";
    Node * root = NULL;
    char * pos = NULL; //For fgets parsing

    //Read a line
    pos = fgets(line,ANITEXTSIZE,fp);
    if(!pos) return NULL; //Null if nothing read
    //Remove any newline
    if ((pos = strchr(line,'\n')) != NULL)
        *pos ='\0';
    //Special case, null node symbol
    if(!strncmp(line,NULLSTRING,ANITEXTSIZE))
        return NULL;

    //Otherwise, add a new node and recurse
    root = (Node *) malloc(sizeof(Node));
    if(!root)
```

```c
	{
		fprintf(stderr, "Error: Unable to allocate memory while loading.\n");
#ifdef _DEBUG
		fflush(stdin); //Make sure there's nothing lurking in the buffer.
		printf("Press Enter to Exit");
		fgetc(stdin);
#endif
		exit(EXIT_FAILURE);
	}
	strncpy(root->text,line,ANITEXTSIZE);
	//And recurse for next nodes
	root->yes = LoadTree(fp);
	root->no = LoadTree(fp);

	return root;
}

//Create some test data
Node * AnimalsTest()
{
	Node * gorilla = (Node *) malloc(sizeof(Node));
	Node * human = (Node *) malloc(sizeof(Node));
	Node * ant = (Node *) malloc(sizeof(Node));
	Node * shark = (Node *) malloc(sizeof(Node));
	Node * mammal = (Node *) malloc(sizeof(Node));
	Node * primate = (Node *) malloc(sizeof(Node));
	Node * nonprimate = (Node *) malloc(sizeof(Node));
	Node * tiger = (Node *) malloc(sizeof(Node));
	Node * nonmammal = (Node *) malloc(sizeof(Node));
	Node * shrew = (Node *) malloc(sizeof(Node));

	Node * root = (Node *) malloc(sizeof(Node));
	strncpy(root->text,"Is it a mammal?", ANITEXTSIZE);
	root->yes = mammal;
	root->no = nonmammal;

	strncpy(mammal->text,"Is it a primate?", ANITEXTSIZE);
	mammal->yes = primate;
	mammal->no = nonprimate;

	strncpy(nonmammal->text,"Is it an insect?", ANITEXTSIZE);
	nonmammal->yes = ant;
	nonmammal->no = shark;

	strncpy(primate->text,"Does it normally wear clothing?", ANITEXTSIZE);
	primate->yes = human;
	primate->no = gorilla;

	strncpy(nonprimate->text,"Is it a predator?", ANITEXTSIZE);
	nonprimate->yes = tiger;
	nonprimate->no = shrew;

	strncpy(ant->text,"Ant", ANITEXTSIZE);
	ant->yes = NULL;
	ant->no = NULL;

	strncpy(human->text,"Human", ANITEXTSIZE);
	human->yes = NULL;
	human->no = NULL;

	strncpy(gorilla->text,"Gorilla", ANITEXTSIZE);
	gorilla->yes = NULL;
	gorilla->no = NULL;

	strncpy(shark->text,"Shark", ANITEXTSIZE);
	shark->yes = NULL;
	shark->no = NULL;

	strncpy(tiger->text,"Tiger", ANITEXTSIZE);
	tiger->yes = NULL;
	tiger->no = NULL;

	strncpy(shrew->text,"Shrew", ANITEXTSIZE);
	shrew->yes = NULL;
	shrew->no = NULL;

	return root;
}
```