```
    1: #include <stdlib.h>
    2: #include <stdio.h>
    3: #include <string.h>
    4: #include "util.h"
    5: #include "animals.h"
    6:
    7: //Play one round of the game (involves
recursive traversal to leaf)
    8: void PlayRound(Node * root)
    9: {
   10:     //Play the node we're at.  Is it a
question or guess node?
   11:     if(root->yes && root->no) //Not a l
eaf, so a question
   12:         if (GetYorN(root->text))
   13:             PlayRound(root->yes);
   14:         else
   15:             PlayRound(root->no);
   16:     else //At leaf.  Need to play leaf
   17:         PlayLeaf(root);
   18: }
   19:
   20: //Handle a leaf node (guess)
   21: void PlayLeaf(Node * leaf)
   22: {
   23:     char szBuff[INPUTBUFFSIZE] = ""; //
Input buffer
   24:     char szName[INPUTBUFFSIZE] = ""; //
Input buffer for animal name
   25:     char szQuestion[INPUTBUFFSIZE] = ""
; //Input buffer for animal question
   26:
   27:     //We're at a leaf, which means gues
s
   28:     sprintf(szBuff,"Is it a(n) %s?",lea
f->text);
   29:     if(GetYorN (szBuff))
   30:     {
   31:         //Succesful guess!
   32:         printf("Yay!\n");
   33:     }
   34:     else
   35:     {
   36:         GetString("Oh, no. What was it?
", szName, INPUTBUFFSIZE);
   37:         sprintf(szBuff,
   38:                 "What is a yes or no qu
estion that would distinguish between \n%s and
 %s?",
   39:                 leaf->text, szName);
   40:         GetString(szBuff,szQuestion,INP
UTBUFFSIZE);
   41:         //Create two new nodes
   42:         leaf->yes = (Node *) malloc(siz
eof(Node));
   43:         leaf->no = (Node *) malloc(size
of(Node));
   44:         //Check that we got our memory
   45:         if(!leaf->yes || !leaf->no)
   46:         {
   47:             fprintf(stderr, "Error: DMA
 Failure.\n");
   48: #ifdef _DEBUG
   49:             fflush(stdin); //Make sure
there's nothing lurking in the buffer.
   50:             printf("Press Enter to Exit
");
   51:             fgetc(stdin);
   52: #endif
   53:             exit(EXIT_FAILURE);
   54:
   55:         }
   56:         //Ensure leaves are terminated
   57:         leaf->yes->yes = leaf->yes->no
= leaf->no->yes = leaf->no->no = NULL;
   58:         sprintf(szBuff,"What would the
answer be for a(n) %s?",leaf->text);
   59:         if(GetYorN(szBuff))
   60:             {
   61:                 strncpy(leaf->yes->text
,leaf->text, ANITEXTSIZE);
   62:                 strncpy(leaf->no->text,
szName, ANITEXTSIZE);
   63:             }
   64:             else
   65:             {
   66:                 strncpy(leaf->no->text,
leaf->text, ANITEXTSIZE);
   67:                 strncpy(leaf->yes->text
,szName, ANITEXTSIZE);
   68:             }
   69:             //Now replace the old leaf
with a new question
   70:         strncpy(leaf->text,CleanStringC
R(szQuestion), ANITEXTSIZE);
   71:             printf("Noted.  Thanks.\n")
;
   72:     }
   73:     return;
   74: }
   75:
   76: //Display the contents of a tree
   77: void PrintTree(Node * root, int indent)
   78: {
   79:     //We'll do a fairly simple preorder
 traversal to print
   80:     //the tree contents, with indenting
 to represent depth.
   81:
   82:     //Ignore NULL nodes
   83:     if(!root) return;
   84:
   85:     //Print this node
   86:     printf("%s\n",root->text);
   87:
   88:     if(root->yes)
   89:     {
   90:         //Indent, print yes branch
   91:         for(int i = 0; i < indent; ++i)
   92:             printf("-");
   93:         printf("-y->");
   94:         PrintTree(root->yes,indent + PR
```

```
INTINDENTSIZE);
    95:     }
    96:
    97:     if(root->no)
    98:     {
    99:         //Indent, print no branch
   100:         for(int i = 0; i < indent; ++i)
   101:             printf("-");
   102:         printf("-n->");
   103:         PrintTree(root->no, indent + PR
INTINDENTSIZE);
   104:     }
   105: }
   106:
   107: //Release memory
   108: Node * Delete(Node * tree)
   109: {
   110:     //Again, we can recursively travers
e our tree to
   111:     //release the memory.
   112:     if(tree->yes)
   113:         tree->yes = Delete(tree->yes);
   114:     if(tree->no)
   115:         tree->no = Delete(tree->no);
   116:     free(tree);
   117:     return NULL;
   118: }
   119:
   120: //Open a file and call SaveTree to
   121: //save tree to it. DESTROYS EXISTING FI
LE
   122: void Save(Node * tree, char * filename)
   123: {
   124:     FILE * fp = fopen(filename,"w");
   125:     if(!fp)
   126:     {
   127:         fprintf(stderr, "Error: Unable
to open file '%s' for writing.\n",
   128:                 filename);
   129: #ifdef _DEBUG
   130:         fflush(stdin); //Make sure ther
e's nothing lurking in the buffer.
   131:         printf("Press Enter to Exit");
   132:         fgetc(stdin);
   133: #endif
   134:         exit(EXIT_FAILURE);
   135:
   136:     }
   137:     SaveTree(tree,fp);
   138:     fclose(fp);
   139: }
   140:
   141: //Recursively save tree to text file
   142: void SaveTree(Node * tree, FILE * fp)
   143: {
   144:     //If NULL, we're at a leaf, write N
ULL symbol and return
   145:     if(!tree)
   146:     {
   147:         fprintf(fp,"%s\n",NULLSTRING);
   148:         return;
   149:     }
   150:     //Otherwise, print node text value
and recurse
   151:     fprintf(fp,"%s\n",tree->text);
   152:     SaveTree(tree->yes,fp);
   153:     SaveTree(tree->no,fp);
   154: }
   155:
   156: //Open a file and call loadtree to
   157: //retrieve a tree
   158: Node * Load(char * filename)
   159: {
   160:     Node * tree = NULL;
   161:     FILE * fp = fopen(filename,"r");
   162:     if(!fp)
   163:     {
   164:         fprintf(stderr, "Error: Unable
to open file '%s' for reading.\n",
   165:                 filename);
   166: #ifdef _DEBUG
   167:         fflush(stdin); //Make sure ther
e's nothing lurking in the buffer.
   168:         printf("Press Enter to Exit");
   169:         fgetc(stdin);
   170: #endif
   171:         exit(EXIT_FAILURE);
   172:     }
   173:     tree = LoadTree(fp);
   174:     fclose(fp);
   175:     return tree;
   176: }
   177:
   178: //Recursively Load tree from a text fil
e
   179: Node * LoadTree(FILE * fp)
   180: {
   181:     //Every line defines a node, in pre
order
   182:     char line[ANITEXTSIZE] = "";
   183:     Node * root = NULL;
   184:     char * pos = NULL; //For fgets pars
ing
   185:
   186:     //Read a line
   187:     pos = fgets(line,ANITEXTSIZE,fp);
   188:     if(!pos) return NULL; //Null if not
hing read
   189:     //Remove any newline
   190:     if ((pos = strchr(line,'\n')) != NU
LL)
   191:         *pos ='\0';
   192:     //Special case, null node symbol
   193:     if(!strncmp(line,NULLSTRING,ANITEXT
SIZE))
   194:         return NULL;
   195:
   196:     //Otherwise, add a new node and rec
urse
   197:     root = (Node *) malloc(sizeof(Node)
);
   198:     if(!root)
```

```
  199:     {
  200:         fprintf(stderr, "Error: Unable
to allocate memory while loading.\n");
  201: #ifdef _DEBUG
  202:         fflush(stdin); //Make sure ther
e's nothing lurking in the buffer.
  203:         printf("Press Enter to Exit");
  204:         fgetc(stdin);
  205: #endif
  206:         exit(EXIT_FAILURE);
  207:     }
  208:     strncpy(root->text,line,ANITEXTSIZE
);
  209:     //And recurse for next nodes
  210:     root->yes = LoadTree(fp);
  211:     root->no = LoadTree(fp);
  212:
  213:     return root;
  214: }
  215:
  216: //Create some test data
  217: Node * AnimalsTest()
  218: {
  219:     Node * gorilla = (Node *) malloc(si
zeof(Node));
  220:     Node * human = (Node *) malloc(size
of(Node));
  221:     Node * ant = (Node *) malloc(sizeof
(Node));
  222:     Node * shark = (Node *) malloc(size
of(Node));
  223:     Node * mammal = (Node *) malloc(siz
eof(Node));
  224:     Node * primate = (Node *) malloc(si
zeof(Node));
  225:     Node * nonprimate = (Node *) malloc
(sizeof(Node));
  226:     Node * tiger = (Node *) malloc(size
of(Node));
  227:     Node * nonmammal = (Node *) malloc(
sizeof(Node));
  228:     Node * shrew = (Node *) malloc(size
of(Node));
  229:
  230:     Node * root = (Node *) malloc(sizeo
f(Node));
  231:     strncpy(root->text,"Is it a mammal?
", ANITEXTSIZE);
  232:     root->yes = mammal;
  233:     root->no = nonmammal;
  234:
  235:     strncpy(mammal->text,"Is it a prima
te?", ANITEXTSIZE);
  236:     mammal->yes = primate;
  237:     mammal->no = nonprimate;
  238:
  239:     strncpy(nonmammal->text,"Is it an i
nsect?", ANITEXTSIZE);
  240:     nonmammal->yes = ant;
  241:     nonmammal->no = shark;
  242:
  243:     strncpy(primate->text,"Does it norm
ally wear clothing?", ANITEXTSIZE);
  244:     primate->yes = human;
  245:     primate->no = gorilla;
  246:
  247:     strncpy(nonprimate->text,"Is it a p
redator?", ANITEXTSIZE);
  248:     nonprimate->yes = tiger;
  249:     nonprimate->no = shrew;
  250:
  251:     strncpy(ant->text,"Ant", ANITEXTSIZ
E);
  252:     ant->yes = NULL;
  253:     ant->no = NULL;
  254:
  255:     strncpy(human->text,"Human", ANITEX
TSIZE);
  256:     human->yes = NULL;
  257:     human->no = NULL;
  258:
  259:     strncpy(gorilla->text,"Gorilla", AN
ITEXTSIZE);
  260:     gorilla->yes = NULL;
  261:     gorilla->no = NULL;
  262:
  263:     strncpy(shark->text,"Shark", ANITEX
TSIZE);
  264:     shark->yes = NULL;
  265:     shark->no = NULL;
  266:
  267:     strncpy(tiger->text,"Tiger", ANITEX
TSIZE);
  268:     tiger->yes = NULL;
  269:     tiger->no = NULL;
  270:
  271:     strncpy(shrew->text,"Shrew", ANITEX
TSIZE);
  272:     shrew->yes = NULL;
  273:     shrew->no = NULL;
  274:
  275:     return root;
  276: }
```