**Background**

This program plays a animal guessing game in which the user is asked a series of yes/no questions about an animal. Eventually, the program makes a guess as to what the animal is. If the program's response is wrong, the user is asked to supply a yes-no question that can distinguish between the program's guess and the correct answer, and this information is added to the program's knowledge base.

The program's internal knowledge base is represented as a binary decision tree, in which internal nodes represent questions and external nodes represent guesses. When the program guesses wrong, the external node containing the guess is replaced by a new sub tree containing the user-supplied distinguishing question, the original guess, and the new entity.

**Methodology:**

The program was written in java and was implemented using tree data structure.Two classes were made for this Lab assignment; the first one being the source code for animal guessing game and the other one is for the Binary Node. It consists of the following:

1. The actual question.
2. The right answer of question or another list of three elements that signifies the question to be asked if the user responded as ‘yes’ for the earlier question.
3. The Wrong answer of the question or another list of three elements that signifies the question to be asked if the user responded as ‘no for the earlier question

Example:

Does it fly? [Pigeon, butterfly]

Does it live underwater? [Fish]

If user gives a yes to this question then it is assumed that the answer could be either a pigeon or a butterfly. If no then, it is assumed to be a fish or dog and further questions are asked. At the end, the user needs to provide a rhetorical question if that animal is not the one that the user have been thinking about.

**Codes:**

**Source code for animal guessing game:**

**Main.cpp**

#include <iostream>

using namespace std;

#include "DecisionTree.h"

bool playMore(); int main(){

cout << "\nWelcome to the game of Animal!\n";

DecisionTree dTree("animal.data"); // load knowledge base

do{

cout << "\nThink of an animal, and I will try to guess it...\n\n";

int winner = dTree.descend();

if (winner)

cout << "\nHa! Even a computer program can beat you...\n"; else

cout << "\nYou were just lucky...\n";

}while ( playMore() );

} // knowledge base is auto-saved to file by DecisionTree destructor

bool playMore(){

char answer;

cout << "\nDo you want to play again (y or n)? "; cin >> answer; return ((answer == 'y') || (answer == 'Y'));

}

**Decision Tree (Header File)**

/\* This file contains the declaration for class DecisionTree.

--------------------------------------------------------------- \*/

#ifndef DECISION\_TREE

#define DECISION\_TREE

#include <iostream> // istream, ostream, <<, >>, ...

#include <string> // string

using namespace std;

class DecisionTree{

public:

DecisionTree(const string& fileName); ~DecisionTree();

int descend();

private:

DecisionTree(const DecisionTree&); // disable copying DecisionTree& operator=(const DecisionTree&); // disable assignment

void read(const string& fileName); void write(const string& fileName);

struct Node

{

Node(const string& question);

Node(const string& question, Node\* leftChild, Node\* rightChild); ~Node();

string myQuestion; // the question I store

Node\* myYesChild; // where to go if the answer is 'y'

Node\* myNoChild; // where to go if the answer is 'n'

};

typedef Node\* NodePtr;

void buildTree(istream& InF, NodePtr& NPtr);

int descendTree(NodePtr& NPtr);

void learnSomething(NodePtr& NPtr);

void writeTree(ostream& OutF, NodePtr NPtr);

NodePtr myRoot;

string myDataFile;

};

inline DecisionTree::DecisionTree(const string& fileName)

{ // just in case fileName is empty

myRoot = 0;

read(fileName); // build myself using data from fileName

myDataFile = fileName; // save for use by destructor

}

inline DecisionTree::~DecisionTree()

{ // write myself back to data file

write(myDataFile);

delete myRoot; // delete root node (recursively deleting everything) // unnecessary, but ...

myRoot = 0;

}

inline DecisionTree::Node::Node(const string& question)

{

myQuestion = question; myYesChild = 0; myNoChild = 0;

}

inline DecisionTree::Node::Node(const string& question, DecisionTree::NodePtr left, DecisionTree::NodePtr right)

{

myQuestion = question; myYesChild = left; myNoChild = right;

}

inline DecisionTree::Node::~Node()

{ myYesChild; // delete yes subtree (recursively)

delete myNoChild; // delete no subtree (recursively)

}

inline int DecisionTree::descend()

{

return descendTree(myRoot);

}

#endif

**Decisiontree.cpp**

/\* This file contains the definition for class DecisionTree.

--------------------------------------------------------------- \*/

#include "DecisionTree.h" // ifstream, ofstream, ...

#include <fstream>

#include <cstdlib> // exit()

using namespace std;

void DecisionTree::read(const string& fileName)

{

ifstream fin( fileName.data() );

if (!fin.is\_open())

{

cerr << "\nDecisionTree::load(fileName) unable to open fileName: '" << fileName << "'; exiting...\n";

exit (-1);

}

buildTree(fin, myRoot);

fin.close();

}

void DecisionTree::buildTree(istream& fin, DecisionTree::NodePtr& nPtr)

{

// trivial case: fin is empty, do nothing if (!fin.eof() && fin.good()) // nontrivial case:

{

string question; // a. read 1 node's data

getline(fin, question);

int leftSubtreeExists, rightSubtreeExists;

fin >> leftSubtreeExists >> rightSubtreeExists; char separator;

fin.get(separator);

// b. build a new node for that data nPtr = new DecisionTree::Node(question);

if (leftSubtreeExists) // c. if necessary

buildTree(fin, nPtr->myYesChild); // build 'yes' node

recursively

if (rightSubtreeExists) // d. if necessary

buildTree(fin, nPtr->myNoChild); // build 'no' node

recursively

}

}

int DecisionTree::descendTree(NodePtr& nPtr)

{

char response; int result;

if (nPtr != 0) // validate parameter

{ // get a y or n response

do

{ // to this node's question

cout << "\n" + nPtr->myQuestion;

cin >> response;

} 'y') && (response != 'n'));

while ((response !=

// if this is a leaf node

if ((nPtr->myYesChild == 0) && (nPtr->myNoChild == 0))

if (response == 'y') // and we guessed correctly

result = 1; // we won!

else // otherwise, we lost, so

{ // learn about the new animal

learnSomething(nPtr);

result = 0;

} // otherwise: it's a non-leaf node

else

if (response == 'y') // so follow the appropriate link

result = descendTree(nPtr->myYesChild); else

result = descendTree(nPtr->myNoChild);

}

return result;

}

void DecisionTree::learnSomething(NodePtr& nPtr)

{

// get new animal name cout << "\nWhat animal are you thinking of? ";

char separator; cin.get(separator); string newAnimal; getline(cin, newAnimal);

// extract old animal name int end = nPtr->myQuestion.find(" (y or n)?", 0);

if (end == string::npos)

{

cerr << "DecisionTree::learnSomething(): ill-formatted question: '" << nPtr->myQuestion << "'; exiting ...\n" << endl;

exit(1);

}

string oldAnimal = nPtr->myQuestion.substr(11, end-10);

// get question to distinguish them

cout << "\nPlease enter a question, such that the answer is\n"<<"\tyes - for a(n) " << newAnimal

<< ", and\n\tno - for a(n) " << oldAnimal << "\n--> ";

string newQuestion; getline(cin, newQuestion);

NodePtr tempPtr = nPtr; // save node containing oldAnimal

nPtr = new Node(newQuestion + " (y or n)? "); new question

// make node for

NodePtr newAnimalPtr = new // make node for newAnimal

Node("\nIs it a(n) " + newAnimal

+ " (y or n)? "); nodes correctly

nPtr->myYesChild = newAnimalPtr; // arrange the 3

nPtr->myNoChild = tempPtr;

}

void DecisionTree::write(const string& fileName)

{

ofstream fout( fileName.data() );

if ( !fout.is\_open() )

{

cerr << "\nDecisonTree::write(fileName): unable to open fileName: '" << fileName << "'; exiting...\n";

exit (-1);

}

writeTree(fout, myRoot);

fout.close();

}

void DecisionTree::writeTree(ostream& fout, DecisionTree::NodePtr nPtr)

{

if (nPtr != 0)

{

fout << nPtr->myQuestion << endl

<< (nPtr->myYesChild != 0) << ' ' << (nPtr->myNoChild != 0) << endl;

writeTree(fout, nPtr->myYesChild); writeTree(fout, nPtr->myNoChild);

}

}

**Animal.cpp**

#include <iostream> #include "DecisionTree.h" using namespace std;

bool playMore();

int main()

{

cout << "\nWelcome to the game of Animal!\n";

DecisionTree dTree("animal.data"); // load tree-data from file

do

{

cout << "\nThink of an animal, and I will try to guess it...\n\n";

int winner = dTree.descend(); // 0 == the person, 1 == the program

if (winner)

cout << "\nHa! Even a computer program can beat you...\n"; else

cout << "\nYou were just lucky...\n";

}

while (playMore());

} // tree-data is auto-saved to file by DecisionTree destructor

bool playMore()

{

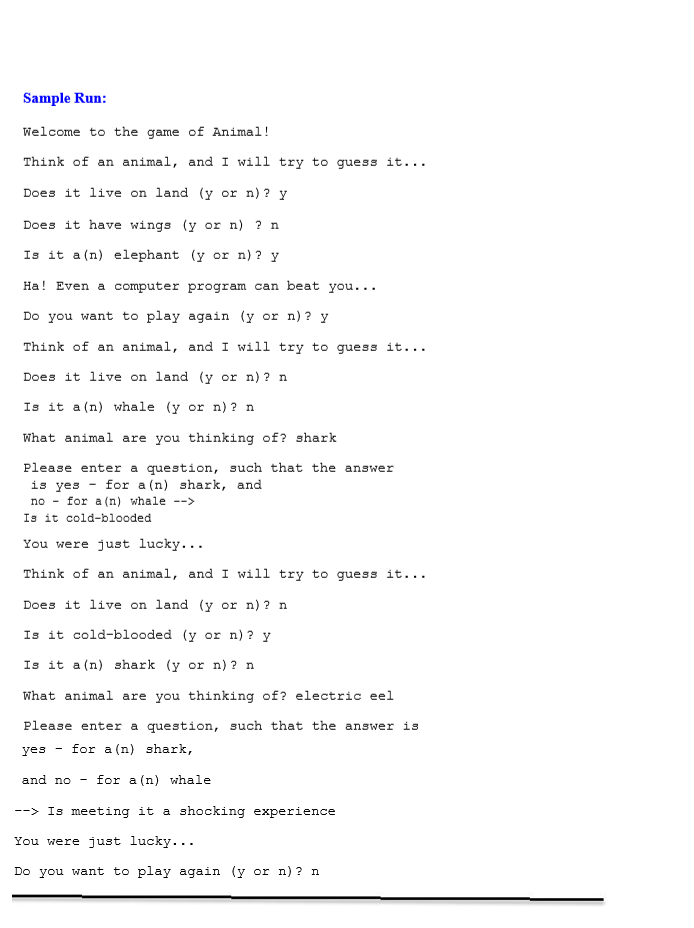
char answer;

cout << "\nDo you want to play again (y or n)? "; cin >> answer;

return ((answer == 'y') || (answer == 'Y'));

}

**Output:**



**Conclusion:**

Hence, Binary tree method was implemented for making the animal guessing game.