CPS222 - DATA STRUCTURES AND ALGORITHMS

An Example of a Program Using Trees: A Simple Guessing Game

(This handout includes both C++ and Java versions)

```
/*
 * GuessingGame.cc
 *
 * This program plays a guessing game in which the user is asked a series
 * of yes/no questions about some unknown entity.  Eventually, the program
 * makes a guess as to what the entity 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 subtree containing the
 * user-supplied distinguishing question, the original guess, and the new
 * entity.
 *
 * At program startup, the user is asked for the name of a data file that
 * contains the program's "knowledge", and at termination the user is given
 * the opportunity to save the updated knowledge base to a file.  The first
 * line in the file describes the program's subject matter, and each subsequent
 * line in the file represents a single node in the tree.  The first word on
 * the line is an integer indicating whether the node is an internal (question)
 * node (1) or an external (guess) node (0); this is followed by a single space;
 * then the remainder of the line is the question or guess, as the case may be.
 * The tree is stored in the file in preorder.
 *
 * Before playing the game, an initial knowledge base file must be created
 * consisting of at least a subject on line 1 and one possible guess on
 * line 2.
 *
 * Copyright (c) 2001, 2003 - Russell C. BJork
 */

#include <string>
#include <iostream>
#include <fstream>
using namespace std;
```

```
class GuessingGame
{
    public:

          // Construct a guessing game instance by reading data from the
          // specified file

          GuessingGame(istream & file);

          // Save a guessing game instance to a file

          void saveTree(ostream & file) const;

          // Play a round of the game - updating tree if final guess is wrong

          void playGame();

          // Destructor - delete all nodes in tree

          ~GuessingGame();

    private:

          string _subject;   // Subject the game is about

          // The game tree is composed of two kinds of nodes - question
          // (internal) nodes and guess (leaf) nodes.  The content of
          // a question node is the question to ask; of a guess node,
          // the answer to propose.  A guess node can be turned into a
          // auestion node when a guess fails

          class Node;

          Node * _root;      // Root of the tree representing the game

          // Recursive auxiliary for constructor

          static Node * readTree(istream & file);

          // Recursive auxiliary for saveTree

          static void writeTree(ostream & file, Node * root);

          // There is no good reason for copying or assigning an
          // object of this class, so by making these private we
          // prevent their inadvertent use and avoid needing to
          // actually implement them

          GuessingGame(const GuessingGame & rhs);
          const GuessingGame & operator = (const GuessingGame & rhs);
};
```

```
class GuessingGame::Node
{
    public:

         // Constructor for a question node - needs question and
         // subtrees to go into if answer is no or yes

         Node(string question, Node * ifNo, Node * ifYes)
         : _isQuestion(true), _contents(question), _lchild(ifNo), _rchild(ifYes)
         { }

         // Constructor for a guess node - needs guess

         Node(string guess)
         : _isQuestion(false), _contents(guess), _lchild(NULL), _rchild(NULL)
         { }

         // Accessors for information stored in a node

         bool isQuestion() const
         { return _isQuestion; }

         string getQuestion() const
         { return _contents; }

         Node * getNoBranch() const
         { return _lchild; }

         Node * getYesBranch() const
         { return _rchild; }

         string getGuess() const
         { return _contents; }

         // Convert a guess node to a question node - needs question
         // and subtrees to go into if answer is no or yes

         void convertToQuestion(string question,
                                Node * ifNo, Node * ifYes)
         {
             _isQuestion = true;
             _contents = question;
             _lchild = ifNo;
             _rchild = ifYes;
         }

         // The destructor recursively deletes any nodes pointed to
         // by this node

         ~Node()
         {
            if (_isQuestion)
            {
                  delete _lchild;
                  delete _rchild;
            }
         }

    private:

         bool _isQuestion;
         string _contents;
         Node * _lchild, * _rchild;
 };
```

```
GuessingGame::GuessingGame(istream & file)
{
    getline(file, _subject);
    _root = readTree(file);
}

GuessingGame::Node * GuessingGame::readTree(istream & file)
{
    // Read the information for this node

    bool isQuestion;
    file >> isQuestion;
    file.get();             // Skip over single blank space
    string contents;
    getline(file, contents);

    // Construct the node, reading subtrees recursively if needed

    if (isQuestion)
    {
        Node * ifNo = readTree(file);
        Node * ifYes = readTree(file);
        return new Node(contents, ifNo, ifYes);
    }
    else
         return new Node(contents);
}

void GuessingGame::saveTree(ostream & file) const
{
    file << _subject << endl;
    writeTree(file, _root);
}

void GuessingGame::writeTree(ostream & file, Node * root)
{
    file << root -> isQuestion() << " ";
    if (root -> isQuestion())
    {
         file << root -> getQuestion() << endl;
        writeTree(file, root -> getNoBranch());
        writeTree(file, root -> getYesBranch());
    }
    else
         file << root -> getGuess() << endl;
}

// Ask the user a yes-no question; return true if user answers yes, false if
// no; reprompt the user if the answer is not recognizable.
bool askYesNo(string question)
{
    string answer;
    do
    {
         // Ask the user the question, read answer, convert to all caps

         cout << question << "? ";
         getline(cin, answer);
         for (int i = 0; i < answer.length(); i ++)
             if (islower(answer[i]))
                  answer[i] = toupper(answer[i]);
         const string YES = "YES";
         const string NO = "NO";

         // Check to see if answer was yes or no.  If so, return appropriate
         // value - else ask again.

         if (answer == YES.substr(0, answer.length()))
             return true;
         else if (answer == NO.substr(0, answer.length()))
             return false;
         else
             cout << "Please answer yes or no" << endl;
    }
    while (true);
}

void GuessingGame::playGame()
{
    Node * current = _root;

    cout << "Please think of a(n) " << _subject << endl;
    if (! askYesNo("Are you thinking of a(n) " + _subject))
         return;

    while (current -> isQuestion())
    {
        if (askYesNo(current -> getQuestion()))
            current = current -> getYesBranch();
        else
            current = current -> getNoBranch();
    }

    if (! askYesNo("Is he/she/it " + current -> getGuess()))
    {
         // Guessed wrong - find out what user was thinking of
         // and get a new question for future use.

         string userAnswer, userQuestion;

         cout << "Who/what were you thinking of? ";
         getline(cin, userAnswer);
         cout << "Please enter a yes/no question that would distinguish "
              << userAnswer << " from " << current -> getGuess() << endl;
         getline(cin, userQuestion);

         // Extend the tree appropriately

         if (askYesNo("For " + userAnswer + " the answer would be"))
             current -> convertToQuestion(userQuestion,
                  new Node(current -> getGuess()), new Node(userAnswer));
         else
             current -> convertToQuestion(userQuestion,
                  new Node(userAnswer), new Node(current -> getGuess()));
    }
}

GuessingGame::~GuessingGame()
{
    delete _root;
}
```

```
// Main program
int main(int argc, char * argv [])
{
    // Access file containing initial knowledge base

    cout << "File to read the knowledge base from? ";
    string filename;
    getline(cin, filename);
    ifstream knowledgeIn(filename.c_str());
    if (! knowledgeIn)
    {
        cerr << "Unable to open file" << endl;
        return 0;
    }

    // Create the game

    GuessingGame theGame(knowledgeIn);
    knowledgeIn.close();

    // Play the game as often as the user wants

    do
    {
        theGame.playGame();
    }
    while (askYesNo("Would you like to play again"));

    // Offer opportunity to save the knowledge base to a file

    cout << "File to write the knowledge base to - blank for none? ";
    getline(cin, filename);

    if (filename.length() > 0)
    {
        ofstream knowledgeOut(filename.c_str());
        if (! knowledgeOut)
        {
            cerr << "Unable to open file" << endl;
            return 0;
        }
        theGame.saveTree(knowledgeOut);
        knowledgeOut.close();
    }
}
```

The same program, but in Java.

(For ease of comparison, this version is directly translated from the C++ version. If written from scratch in Java, it might well use a GUI rather than a console interface, and might also use a slightly different knowledge file format. This version works with the same knowledge files as the C++ version, too.)

```
/*
 * GuessingGame.java
 *
   ... REMAINDER OF PROLOGUE COMMENT SAME AS C++ - OMITTED TO CONSERVE PAPER

 */

import java.io.*;

/** An object of this class represents a guessing game. */

public class GuessingGame
{
    /** Constructor
     *
     *  @param file the file to read the game tree from
     */
    public GuessingGame(BufferedReader file) throws IOException
    {
        subject = file.readLine();
        root = readTree(file);
    }

    /** Save the (possibly modified) game tree to a file
     *
     *  @param file the file to save the game tree to
     */
    public void saveTree(PrintWriter file) throws IOException
    {
        file.println(subject);
        writeTree(file, root);
    }

    /** Play an instance of the game, updating the tree if needed
     */
    public void playGame() throws IOException
    {
        Node current = root;

        System.out.println("Please think of a(n) " + subject);
        if (! askYesNo("Are you thinking of a(n) " + subject))
            return;

        while (current.isQuestion())
        {
            if (askYesNo(current.getQuestion()))
                current = current.getYesBranch();
            else
                current = current.getNoBranch();
        }
        if (! askYesNo("Is he/she/it " + current.getGuess()))
        {
            // Guessed wrong - find out what user was thinking of
            // and get a new question for future use.

            String userAnswer, userQuestion;

            System.out.print("Who/what were you thinking of? ");
            userAnswer = consoleIn.readLine();
            System.out.println(
                  "Please enter a yes/no question that would distinguish "
                + userAnswer + " from " + current.getGuess());
            userQuestion = consoleIn.readLine();

            // Extend the tree appropriately

            if (askYesNo("For " + userAnswer + " the answer would be"))
                current.convertToQuestion(userQuestion,
                    new Node(current.getGuess()), new Node(userAnswer));
            else
                current.convertToQuestion(userQuestion,
                    new Node(userAnswer), new Node(current.getGuess()));
        }
    }

    /** Main program */

    public static void main(String[] args) throws IOException
    {
        // Access file containing initial knowledge base

        System.out.print("File to read the knowledge base from? ");
        String filename;
        filename = consoleIn.readLine();
        BufferedReader knowledgeIn =
            new BufferedReader(new FileReader(filename));

        // Create the game

        GuessingGame theGame = new GuessingGame(knowledgeIn);
        knowledgeIn.close();

        // Play the game as often as the user wants

        do
        {
            theGame.playGame();
        }
        while (askYesNo("Would you like to play again"));

        // Offer opportunity to save the knowledge base to a file

        System.out.print("File to write the knowledge base to - blank for none? ");
        filename = consoleIn.readLine();

        if (filename.length() > 0)
        {
            PrintWriter knowledgeOut = new PrintWriter(new FileWriter(filename));
            theGame.saveTree(knowledgeOut);
            knowledgeOut.close();
        }

        System.exit(0);
    }
```

```
/* Instance variables */

private String subject;     // Subject the game is about

private Node root;          // Root of internal knowledge tree

/* Private methods - auxiliary to public methods above */

/** Read a tree stored in preorder in a file
 *
 *  @param file the file to read from
 *  @return root of resultant tree
 */

private static Node readTree(BufferedReader file) throws IOException
{
    // Read the information for this node

    boolean isQuestion = ((char) file.read() == '1');
    file.skip(1);       // Skip over single blank space
    String contents = file.readLine();

    // Construct the node, reading subtrees recursively if needed

    if (isQuestion)
    {
        Node ifNo = readTree(file);
        Node ifYes = readTree(file);
        return new Node(contents, ifNo, ifYes);
    }
    else
        return new Node(contents);
}

/** Write a tree to a file in preorder
 *
 *  @param file the file to write to
 *  @param root the root of the tree
 */

private static void writeTree(PrintWriter file, Node root) throws IOException
{
    file.print(root.isQuestion() ? 1 : 0);
    file.print(" ");
    if (root.isQuestion())
    {
        file.println(root.getQuestion());
        writeTree(file, root.getNoBranch());
        writeTree(file, root.getYesBranch());
    }
    else
        file.println(root.getGuess());
}
```

```
/** Ask the user a yes-no question
 *
 *  @param question the question to ask
 *  @returntrue if user answers yes, false if no
 *
 *  (reprompt the user if the answer is not recognizable.)
 */

private static boolean askYesNo(String question) throws IOException
{
    String answer;
    do
    {
        // Ask the user the question, read answer, convert to all caps

        System.out.print(question + "? ");
        answer = consoleIn.readLine();

        // Check to see if answer was yes or no.  If so, return appropriate
        // value - else ask again.

        if (answer.equalsIgnoreCase("YES".substring(0, answer.length())))
            return true;
        else if (answer.equalsIgnoreCase("NO".substring(0, answer.length())))
            return false;
        else
            System.out.println("Please answer yes or no");
    }
    while (true);
}

/* Wrap System.in in a BufferedReader object so we can use readLine(),
 * etc. on it.
 */

private static BufferedReader consoleIn =
    new BufferedReader(new InputStreamReader(System.in));
```

```
/** The game tree is composed of two kinds of nodes - question
 *  (internal) nodes and guess (leaf) nodes.  The content of
 *  a question node is the question to ask; of a guess node,
 *  the answer to propose.  A guess node can be turned into a
 *  question node when a guess fails
 */

private static class Node
{
    /** Constructor for a question node
     *
     *  @param question the question to ask
     *  @param ifNo the subtree to go into if user answers no
     *  @param ifYes the subtree to go into if user answers yes
     */

    Node(String question, Node ifNo, Node ifYes)
    {
        isQuestion = true;
        contents = question;
        this.lchild = ifNo;
        this.rchild = ifYes;
    }

    /** Constructor for a guess node
     *
     *  @param guess the guess to try
     */

    Node(String guess)
    {
        isQuestion = false;
        contents = guess;
        lchild = null;
        rchild = null;
    }

    /** Accessor for whether a node represents a question or a guess
     *
     *  @return true if a question, false if a guess
     */

    boolean isQuestion()
    {
        return isQuestion;
    }

    /** Accessor for question stored in a node.
     *  Precondition: the node represents a question
     *
     *  @return the question stored
     */

    String getQuestion()
    {
        return contents;
    }
```

```
        /** Accessor for "no" branch from a question node.
         *  Precondition: the node represents a question
         *
         *  @return root of the "no" branch
         */

        Node getNoBranch()
        {
            return lchild;
        }

        /** Accessor for "yes" branch from a question node.
         *  Precondition: the node represents a question
         *
         *  @return root of the "yes" branch
         */

        Node getYesBranch()
        {
            return rchild;
        }

        /** Accessor for guess stored in a node.
         *  Precondition: the node represents a guess
         *
         *  @return the guess stored
         */

        String getGuess()
        {
            return contents;
        }

        /** Convert a guess node to a question node
         *  Precondition: the node currently represents a guess
         *
         *  @param question the question to ask
         *  @param ifNo the subtree to go into if user answers no
         *  @param ifYes the subtree to go into if user answers yes
         */

        void convertToQuestion(String question, Node ifNo, Node ifYes)
        {
            isQuestion = true;
            contents = question;
            lchild = ifNo;
            rchild = ifYes;
        }

        /* Instance variables of a Node */

        private boolean isQuestion; // True for question, false for guess
        private String contents;    // Question or quess as the case may be
        private Node lchild, rchild;// "No" and "Yes" branches for a question
    }
}
```