**Deerwalk Institute of Technology  
*Sifal, Kathmandu***

LAB #1

**Submitted By:**

Ruby Shrestha

Roll No. 0341

Section A

**Submitted To:**

Birodh Risal

Artificial Intelligence

DWIT

Date: 12 May 2016

Contents

[Problem Definition 3](#_Toc450849572)

[Methodology 3](#_Toc450849573)

[Data Structure 3](#_Toc450849574)

[Algorithm and Program Flow 4](#_Toc450849575)

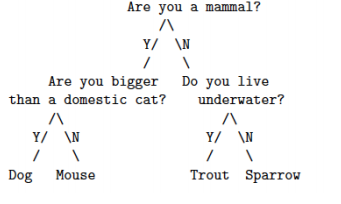
[Implementation / Program 4](#_Toc450849576)

[Outputs 7](#_Toc450849577)

[Analysis of Output 8](#_Toc450849578)

# Problem Definition

The goal of the program to be developed is to implement an “Animal Guessing Game”. In this game, the player thinks of an animal and the computer tries to guess which animal the player is imagining by asking questions which can be answered `yes' or `no'. To begin each round of the game, the user needs to think of an animal. For example, suppose dog is the animal that the user is thinking of. The computer will then generate the question at the root node “Are you a mammal?" as the first question. The user has to press 'y' or 'n' (1 or 0) as the response. Based on the response the computer will generate the second question. The game continues until the leaf node is reached. At the leaf node the computer will generate the guess (dog, mouse, trout or sparrow in the example considered).



# Methodology

## Data Structure

In order to program the Animal Guessing Game, **linked list** has been considered as the preferred data structure and has been implemented using Java as programming language. The nodes of linked list have been implemented as instances of Node class.

**public class** Node {  
 Node **left**; //represents the left child, which itself is a node  
 Node **right**; //represents the right child, which itself is a node  
 String **str**; //str is the data that is present in the node  
 **int weight**;

}

Note: **weight** is an integer value which helps to determine whether this node shall be a left or a right child of the parent node.

String Array has been used to store the data that shall be input into the nodes of the linked list as follows:

String[] str={**"Dog"**, **"Are you a bigger than a domestic cat?"**,**"Mouse"**, **"Are you a mammal?"**, **"Trout"**, **"Do you live underwater?"**, **"Sparrow"**};

Note: The trick used here is that the middle element of the array has the root, the array elements to the left give the left sub-tree and the array elements to the right give the right sub-tree. Same follows for the sub-trees.

## Algorithm and Program Flow

1. Start
2. Store all the data to be input into the linked list in such a way that you are following the trick mentioned above.
3. Add the data of the array as nodes of the linked list following a recursive function call technique. That is, starting by adding the middle index array value as node of the linked list, then calling the recursive function for elements to the left of the middle index and doing the same for the elements to the right of the middle index, continuing till start index != end index.
4. After linked list is formed, the game is started. Until the leaf node is reached, the questions (data in nodes) are asked and depending upon the response (yes/no), traversal is done to the left or right sub-tree.
5. Final response is given based on which leaf node is reached through traversal.
6. End

## Implementation / Program

*The functions developed and used to implement the problem (Animal Guessing Game) are described in comments.*

**Node Class**

*/\*\*  
 \* Created by Ruby Shrestha | 0341 | Section A on 3/30/2016.  
 \*/***public class** Node {  
 Node **left**;  
 Node **right**;  
 String **str**;  
 **int weight**;  
}

**Game Implementation [Class Name – BinaryImplementation]**

**import** java.util.\*;  
  
*/\*\*  
 \* Created by Ruby Shrestha | 0341 | Section A on 3/30/2016.  
 \*/***public class** BinaryTreeImplementation {  
  
 *//main section : code execution starts here* **public static void** main(String[] args) {  
 Node n=**null**;  
 *//elements of str will be input into tree* String[] str={**"Dog"**, **"Are you a bigger than a domestic cat?"**,**"Mouse"**, **"Are you a mammal?"**, **"Trout"**, **"Do you live underwater?"**, **"Sparrow"**};  
 n=*addToTree*(str, 0, str.**length**-1, n);  
 *playGame*(n);  
 }

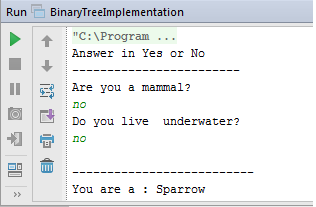
*/\* addToTree method is implemented as a recursive function. In each call, the middle index of the array is chosen and inserted as  
 \* node into the binary tree, and two recursive calls to the same function are made: one considering left half from the middle index  
 \* and next, considering right half of the array from the middle index. Though same array is passed in every call, start and end values  
 \* make it appear as if we are passing the array after dividing it into halves in evry call. The concept of division used here is similar to that  
 \* of binary search.  
 \* The process is repeated until start==end which means only one element is left and it cannot be divided anymore into halves.\*/*

**static** Node addToTree(String[] arr, **int** start, **int** end, Node n){  
 **int** mid=(start+end)/2;  
 n=*addNode*(n, mid, arr[mid]);  
 **if** (start!=end){  
 *addToTree*(arr, start, mid - 1, n);  
 *addToTree*(arr, mid+1, end, n);  
 }  
 **return** n;  
 }  
  
 */\*addNode method simply adds node in the correct position based on weight of the node. If the weight of the node to be  
 \* added is less than the root, we move to the left subtree else we move to the right subtree. We do so until leaf node is reached.  
 \* The weight of the node to be added is then compared with the weight of the leaf node and added in correct direction (left/right).  
 \* Since java does not pass parameter by reference, this function returns the updated node in order to make the changes made to the node passed as  
 \* parameter visible in the calling function.\*/* **static** Node addNode(Node n, **int** weight, String str){  
 Node newNode=**new** Node();  
  
 *//In case the tree does not exit, n==null. In that situation, we need to create root* **if** (n==**null**){  
 newNode.**left**=**null**;  
 newNode.**right**=**null**;  
 newNode.**str**=str;  
 newNode.**weight**=weight;  
 n=newNode;  
 }**else**{ *//for other cases besides root* newNode.**str**=str;  
 newNode.**left**=**null**;  
 newNode.**right**=**null**;  
 newNode.**weight**=weight;  
 **while** (!*isLeafNode*(n)){  
 **if** (weight<n.**weight**){  
 **if** (n.**left**!=**null**){  
 n=n.**left**;  
 }**else**{  
 **break**;  
 }  
 }**else**{  
 **if** (n.**right**!=**null**){  
 n=n.**right**;  
 }**else**{  
 **break**;  
 }  
 }  
 }  
 **if** (weight<n.**weight**){  
 n.**left**=newNode;  
 }**else** {  
 n.**right**=newNode;  
 }  
 }  
 **return** n;  
 }

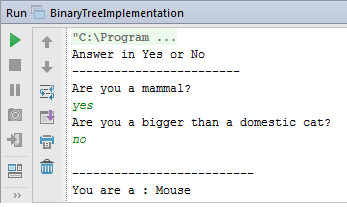
*/\*Implements the game. If the user replies as YES to the question asked, move to left subtree else to right.\*/* **static void** playGame(Node n){  
 Scanner usInput= **new** Scanner(System.***in***);  
 System.***out***.println(**"Answer in Yes or No\n------------------------"**);  
 **while** (!*isLeafNode*(n)){  
 System.***out***.println(n.**str**);  
 String reply=usInput.nextLine().toLowerCase();  
 **if** (reply.equals(**"yes"**)){  
 n=n.**left**;  
 }**else** {  
 n=n.**right**;  
 }  
 }  
 System.***out***.println(**"\n--------------------------\nYou are a : "** + n.**str**);  
 }

*/\*Checks if the node passed as parameter is the leaf node\*/* **static boolean** isLeafNode(Node n){  
 **return** (n.**left**==**null** && n.**right**==**null**);  
 }  
}

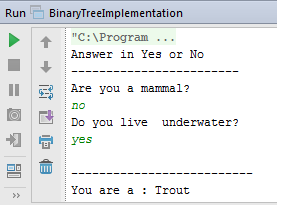
# Outputs



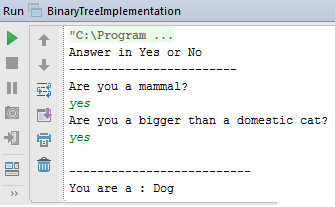
Example 1



Example 2



Example 3



Example 4

## Analysis of Output

Let us consider Example 1

Questions are asked starting with the data/question in the root node: “Are you a mammal?” According to the problem definition, we need to traverse to the right sub-tree on every No and to the left sub-tree on every Yes. Since the user says No, we move to the right sub-tree. The question on the root of the right sub-tree is then asked: “Do you live under water?” The user again says No, so we move to the right sub-tree. We reach the leaf node. Traversal stops when the leaf node is reached. The result here then becomes: “Sparrow”.