Class	CMPS 261
Section	001
Problem	Programming Assignment #1
Name	McKelvy, James Markus
CLID	Jmm0468
Due Date	12:30pm September 22, 2005

II. Design Documentation

II.1 System Architecture Description

The object(s) within the program are:
BinaryTree : BinaryTree.h

The main objective of the application is to utilize the implementation of the BinaryTree class. The application's main driver will be used to test the overall stability and integrity of the implementation of the BinaryTree class by testing its creation and its various member methods including traversals. The main driver will create three types of binary trees: an empty binary tree; a binary tree with only one node; and a binary tree that has a root and two subtrees that are referred to as leftTree and rightTree. The BinaryTree class will create BinaryTree objects will that interact with each other in a way that some BinaryTrees will be parents, and others children of each other.

II.2 Information about the Objects

II.2.1 Class Information
Name: BinaryTree
Description: Simulates a mathematical binary tree, allows for traversals, and emptiness checking.
Base Class: N/A

II.2.2 Class Attributes
Name: root
Description: The root of this binary tree. Contains information to two subtrees (possibly).

Contains information about itself (this node).
Type: Node
Range of acceptable values: Any value that corresponds to the template class NodeType.

II.2.3 Class Operations
Prototype: void inOrder();
Description: Performs an inorder traversal of this tree, outputting the value at the root node.
Precondition: The binary tree must be initialized.
Postcondition: N/A
Cost Analysis: O(n)

Visibility: public

Prototype: void preOrder();
Description: Performs a preorder traversal of this tree, outputting the value at the root node.
Precondition: The binary tree must be initialized.
Postcondition: N/A
Cost Analysis: O(n)
Visibility: public

Prototype: void postOrder();
Description: Performs a postorder traversal of this tree, outputting the value at the root node.
Precondition: The binary tree must be initialized.
Postcondition: N/A
Cost Analysis: O(n)
Visibility: public

Prototype: bool isEmpty();
Description: Helps to determine if there are any values stored in the binary tree.
Precondition: The binary tree must be initialized.
Postcondition: Returns true if the tree is empty, false if the tree is not empty.
Cost Analysis: O(n)
Visibility: public

II.3 Information about the Main Application

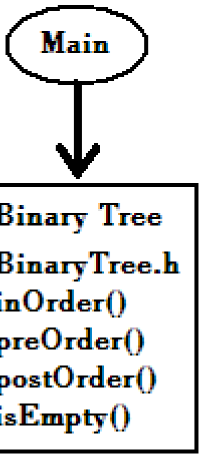
```
#include<iostream>
#include<cstdlib>
#include<string>
#include "BinaryTree.h"

using namespace std;

void pause();

int main(){
   cout << "Creating an empty binary tree." << endl;
   BinaryTree<char> * tree1 = new BinaryTree<char>();
   cout << "Is tree1 empty? ";
   if(tree1->isEmpty())
     cout << "Yes." << endl;
   else
     cout << "No." << endl;
   cout << endl << "Performing inorder traversal. " << endl;
   tree1->inOrder();
```

```
cout << endl << "Performing preorder traversal. " << endl;
tree1->preOrder();
cout << endl << "Performing postorder traversal. " << endl;
tree1->postOrder();
delete tree1;

pause();


cout << endl << "Creating a non-empty binary tree." << endl;
BinaryTree<char> * tree2 = new BinaryTree<char>('2');
cout << "Is tree2 empty? ";
if(tree2->isEmpty())
  cout << "Yes." << endl;
else
  cout << "No." << endl;
delete tree2;

pause();

cout << endl << "Creating a binary tree with only one node." << endl;
BinaryTree<char> * tree3 = new BinaryTree<char>('3');
cout << endl << "Performing inorder traversal. " << endl;
tree3->inOrder();
cout << endl << "Performing preorder traversal. " << endl;
tree3->preOrder();
cout << endl << "Performing postorder traversal. " << endl;
tree3->postOrder();
cout << endl;
delete tree3;

pause();

cout << endl << "Creating a populated binary tree." << endl;
cout << "       root" << endl;
cout << "      /    \\" << endl;
cout << "    l1      r1" << endl;
cout << "   /  \\    /  \\" << endl;
cout << "ll2  lr2  rl2  rr2" << endl;
cout << endl;
BinaryTree<string> * ll2 = new BinaryTree<string>("ll2", NULL, NULL);
BinaryTree<string> * lr2 = new BinaryTree<string>("lr2", NULL, NULL);
BinaryTree<string> * l1 = new BinaryTree<string>("l1", ll2, lr2);

BinaryTree<string> * rl2 = new BinaryTree<string>("rl2", NULL, NULL);
BinaryTree<string> * rr2 = new BinaryTree<string>("rr2", NULL, NULL);
```

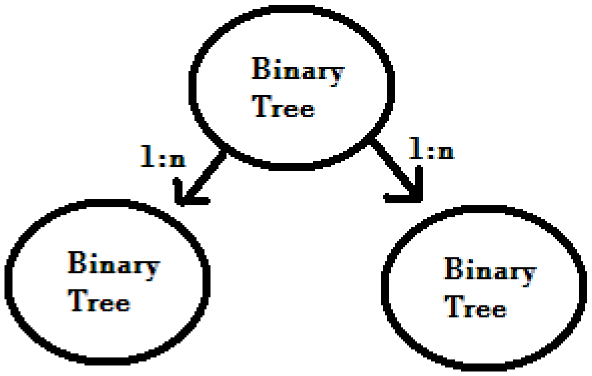
```
    BinaryTree<string> * r1 = new BinaryTree<string>("r1", rl2, rl2);

    BinaryTree<string> * root = new BinaryTree<string>("root", l1, r1);

    cout << "Is root empty? ";
    if(root->isEmpty())
      cout << "Yes." << endl;
    else
      cout << "No." << endl;
    cout << endl << "Performing inorder traversal. " << endl;
    root->inOrder();
    cout << endl << "Performing preorder traversal. " << endl;
    root->preOrder();
    cout << endl << "Performing postorder traversal. " << endl;
    root->postOrder();
    cout << endl;
    delete root;

    pause();

    return 0;
}

void pause(){
  cout << "Press ENTER.";
  getchar();
  cout << endl;
}
```

II.4 Design Diagrams

II.4.1 Object Interaction Diagram



II.4.2 Aggregation Diagram


Binary Tree
1:n
1:n
Binary Tree
Binary Tree