05/06/17 02:18:38 /Users/hostname/Desktop/CSE 330/Lab5/[Lab5] Adnar Lozano.cpp

```
 1  // Adnar Lozano
 2  // CSE 330 Data Structures
 3  // Lab 5 (Trees)
 4  // 5/4/17
 5
 6  #include <iostream>
 7  #include <queue>
 8  using namespace std;
 9  class Node {
10  public:
11      int data;
12      Node* left;
13      Node* right;
14  };
15  Node* GetNewNode(int data) {
16      Node* newNode = new Node();
17      newNode->data = data;
18      newNode->left = NULL;
19      newNode->right = NULL;
20      return newNode;
21  }
22  Node* Insert(Node* root, int data) {
23      if (root == NULL) {
24          root = GetNewNode(data);
25      }
26      else if (data <= root->data)
27          root->left = Insert(root->left, data);
28      else root->right = Insert(root->right, data);
29      return root;
30  }
31  bool Search(Node* root, int data) {
32      if(root == NULL)
33          return false;
34      else if (root->data == data)
35          return true;
36      else if (data <= root->data)
37          return Search(root->left, data);
38      else return Search(root->right, data);
39  }
40  // Recursive Function
41  int FindMin(Node* root) {
42      if (root == NULL) {
43          cout << "Error: Tree is empty\n";
44          return -1;
45      }
46      while (root->left == NULL)
47          return root ->data;
48      return FindMin(root->left);
49  }
50  // Iterative function
51  int FindMax(Node* root) {
52      if (root == NULL) {
53          cout << "Error: Tree is empty\n";
54          return -1;
55      }
56      while (root->right != NULL)
57          root = root->right;
58      return root->data;
59  }
60  int FindHeight(Node* root) {
61      if (root == NULL) return -1;
62      return max(FindHeight(root->left), FindHeight(root->right))+1;
63  }
64  void LevelOrder(Node* root) {
65      if(root == NULL) return;
66      queue<Node*> Q;
67      Q.push(root);
68       while(!Q.empty()) {
```

```
69          Node* root = Q.front();
70          cout << root->data << " ";
71          if(root->left != NULL) Q.push(root->left);
72          if(root->right != NULL) Q.push(root->right);
73          Q.pop();
74      }
75  }
76  void PreOrder(Node* root) {
77      if(root == NULL) return;
78      cout << root->data << " ";
79      PreOrder(root->left);
80      PreOrder(root->right);
81  }
82  void InOrder(Node* root) {
83      if(root == NULL) return;
84      InOrder(root->left);
85      cout << root->data << " ";
86      InOrder(root->right);
87  }
88  void PostOrder(Node* root) {
89      if(root == NULL) return;
90      PostOrder(root->left);
91      PostOrder(root->right);
92      cout << root->data << " ";
93  }
94  bool IsBST(Node* root) {
95      if(root == NULL) return true;
96      if(root->data > INT_MIN && root->data < INT_MAX
97          && IsBST(root->left)
98          && IsBST(root->right))
99          return true;
100     else return false;
101 }
102 int main() {
103     Node* root = NULL;
104     root = Insert(root,15);
105     root = Insert(root,10);
106     root = Insert(root,20);
107     root = Insert(root,12);
108     root = Insert(root,25);
109     root = Insert(root,5);
110     root = Insert(root,35);
111     int number;
112     cout << "Enter number to search: ";
113     cin >> number;
114     cout << number << endl;
115     if (Search(root,number) == true) cout << "Number was Found\n";
116     else cout << "Number was Not found\n";
117     cout << "Root: " << root->data << endl;
118     cout << "Min value is: " << FindMin(root) << endl;
119     cout << "Max value is: " << FindMax(root) << endl;
120     cout << "Height is: " << FindHeight(root) << endl;
121     cout << "LevelOrder: ";
122     LevelOrder(root);
123     cout << endl;
124     cout << "PreOrder: ";
125     PreOrder(root);
126     cout << endl;
127     cout << "InOrder: ";
128     InOrder(root);
129     cout << endl;
130     cout << "PostOrder: ";
131     PostOrder(root);
132     cout << endl;
133     if (IsBST(root) == true) cout << "It's a Binary Tree\n";
134     else cout << "Not a Binary Tree\n";
135     return 0;
136 }
```