DS LAB 10

Task 1:

Code

```
#include<iostream>
using namespace std;
class Node{
public:
  int data;
  Node *left, *right;
  Node(int val): data(val), left(nullptr), right(nullptr) {}
};
class Tree{
private:
  Node *root;
  Node* insertRec(Node *root,int val) {
    if(root == nullptr) return new Node(val);
    if(val < root->data) root->left = insertRec(root->left, val);

    else if(val > root->data) root->right = insertRec(root->right,val);

    return root;
  }

  void inOrder(Node* root) {
    if(root == nullptr) return;

    inOrder(root->left);
    cout<<root->data<<" ";
    inOrder(root->right);
  }

  int findMinimum(Node* root) {
    while(root != nullptr && root->left != nullptr) root = root->left;

    return root->data;
  }

  int findMaximum(Node* root) {
    while(root != nullptr && root->right != nullptr) root = root->right;
```

```
        return root->data;
    }
public:
    Tree() : root(nullptr) {}
    void insert(int val) {
        root = insertRec(root,val);
    }

    void display() {
        inOrder(root);
        cout<<endl;
    }

    void getMinimum() { cout<<findMinimum(root)<<endl;}
    void getMaximum() { cout<<findMaximum(root)<<endl;}
};
int main() {
    Tree t1;

    t1.insert(50);
    t1.insert(60);
    t1.insert(30);
    t1.insert(55);
    t1.insert(35);
    t1.insert(10);
    t1.insert(65);

    t1.display();

    t1.getMinimum();
    t1.getMaximum();

    // cout<<"Maximum Value is: "<<t1.getMaximum()<<endl;

    return 0;
}
```

Task 2:

Code

```
#include<iostream>
using namespace std;
class Node{
public:
  int data;
  Node *left, *right;
  Node(int val): data(val), left(nullptr), right(nullptr) {}
};
class Tree{
private:
  Node *root;
  Node* insertRec(Node *root,int val) {
    if(root == nullptr) return new Node(val);
    if(val < root->data) root->left = insertRec(root->left, val);

    else if(val > root->data) root->right = insertRec(root->right,val);

    return root;
  }

  void inOrder(Node* root) {
    if(root == nullptr) return;

    inOrder(root->left);
    cout<<root->data<<" ";
    inOrder(root->right);
  }

  Node* search(Node*root, int target) {
    if(root == nullptr || root->data == target) return root;
    if(target < root->data) return search(root->left, target);
    else if(target > root->data) return search(root->right,target);
  }

  Node* findPredecessor(int target) {
    Node *pred = nullptr;
    Node *curr = root;
```

```
    while(curr != nullptr) {
      if(target > curr->data) {
        pred = curr;
        curr = curr->right;
      } else curr = curr->left;
    }
    return pred;
  }

  Node* findSuccessor(int target) {
    Node *curr = root;
    Node* successor  = nullptr;

    while(curr != nullptr) {
      if(target < curr->data) {
        successor = curr;
        curr = curr->left;
      } else curr = curr->right;
    }
    return successor;
  }
public:
  Tree() : root(nullptr) {}
  void insert(int val) {
    root = insertRec(root,val);
  }

  void findPred_Succ(int target) {
    Node *pred = findPredecessor(target);
    if(!pred) cout<<"Predecessor not found"<<endl;
    else cout<<"predecessor = "<<pred->data<<endl;


    Node *suc = findSuccessor(target);
    if(!suc) cout<<"Successor not found"<<endl;
    else cout<<"Successor = "<<suc->data<<endl;
  }


  void display() {
    inOrder(root);
    cout<<endl;
  }
```

```
};
int main() {
   Tree t1;

   t1.insert(50);
   t1.insert(60);
   t1.insert(30);
   t1.insert(55);
   t1.insert(35);
   t1.insert(10);
   t1.insert(65);

   t1.display();
   t1.findPred_Succ(50);


   return 0;
}
```

Task 3:

Code

```
#include<iostream>
using namespace std;
class Node{
public:
   int data;
   Node *left, *right;
   Node(int val): data(val), left(nullptr), right(nullptr) {}
};
class Tree{
private:
   Node *root;
   Node* insertRec(Node *root,int val) {
      if(root == nullptr) return new Node(val);
      if(val < root->data) root->left = insertRec(root->left, val);

      else if(val > root->data) root->right = insertRec(root->right,val);

      return root;
   }
```

```
  void inOrder(Node* root) {
    if(root == nullptr) return;

    inOrder(root->left);
    cout<<root->data<<" ";
    inOrder(root->right);
  }
  int countNodes(Node* root) {
    if (root == nullptr) return 0;
    return 1 + countNodes(root->left) + countNodes(root->right);
  }
public:
  Tree() : root(nullptr) {}
  void insert(int val) {
    root = insertRec(root,val);
  }

  void checkBalance() {
    if(root == nullptr) {
      cout<<"Tree is empty."<<endl;
      return;
    }

    int leftCount = countNodes(root->left);
    int rightCount = countNodes(root->right);

    if(leftCount > rightCount) cout<<"The tree has more nodes on the left side."<<endl;
    else if(rightCount > leftCount) cout<<"The tree has more nodes on the right
side."<<endl;

    else cout<<"The tree has equal nodes on both sides."<<endl;
  }

  void display() {
    inOrder(root);
    cout<<endl;
  }

};
int main() {
  Tree t1;

  t1.insert(10);
```

```
    t1.insert(5);
    t1.insert(15);
    t1.insert(3);
    t1.insert(7);
    t1.insert(12);
    // t1.insert(18);
    // t1.insert(17);
    // t1.insert(20);

    t1.display();

    t1.checkBalance();
    return 0;
}
```

Task 4:

Code

```
#include<iostream>
using namespace std;
class Node{
public:
    int data;
    Node *left, *right;
    Node(int val): data(val), left(nullptr), right(nullptr) {}
};
class Tree{
private:
    Node *root;
    Node* insertRec(Node *root,int val) {
        if(root == nullptr) return new Node(val);
        if(val < root->data) root->left = insertRec(root->left, val);

        else if(val > root->data) root->right = insertRec(root->right,val);

        return root;
    }

    void inOrder(Node* root) {
        if(root == nullptr) return;

        inOrder(root->left);
```

```
    cout<<root->data<<" ";
    inOrder(root->right);
  }
public:
  Tree() : root(nullptr) {}
  void insert(int val) {
    root = insertRec(root,val);
  }

  void insertAfterVal(int val, int newVal) {
    Node* curr = root;
    while(curr != nullptr) {
      if(curr->data == val) {
        Node* newNode = new Node(newVal);
        if(curr->right == nullptr) {
          curr->right = newNode;
        } else {
          newNode->right = curr->right;
          curr->right = newNode;
        }
        return;
      } else if(val < curr->data) {
        curr = curr->left;
      } else {
        curr = curr->right;
      }
    }
    cout<<"Value "<<val<<" not found in the tree."<<endl;
  }

  void display() {
    inOrder(root);
    cout<<endl;
  }

};
int main() {
  Tree t1;

  t1.insert(10);
  t1.insert(5);
  t1.insert(15);
  t1.insert(3);
  t1.insert(7);
```

```
    t1.insert(12);
    // t1.insert(18);
    // t1.insert(17);
    // t1.insert(20);

    t1.insertAfterVal(5, 8);

    t1.display();
    return 0;
}
```