Name : Faricha Aulia

Class : 1I - IT

Absen : 08

```
package BinarySearchTree;

public class BinaryTree08 {
   Node08 root;

   public BinaryTree08(){
      root = null;
   }
   boolean isEmpty(){
      return root==null;
   }

   void add(int data){
      if(isEmpty()){
         root = new Node08(data);
      } else {
         Node08 current = root;
         while(true){
            if(data<current.data){
               if(current.left!=null){
                  current = current.left;
               }else{
                  current.left = new Node08(data);
                  break;
               }
            }else if(data>current.data){
               if(current.right!=null){
                  current = current.right;
               }else{
                  current.right = new Node08(data);
                  break;
               }
            }else{
               break;
            }
         }
      }
   }
```

```
boolean find(int data){
    boolean result = false;
    Node08 current = root;
    while(current!=null){
        if(current.data==data){
            result = true;
            break;
        }else if(data<current.data){
            current = current.left;
        }else{
            current = current.right;
        }
    }
    return result;
}

void traversePreOrder(Node08 node){
    if(node!=null){
        System.out.println("" + node.data);
        traversePreOrder(node.left);
        traversePreOrder(node.right);
    }
}

void traversePostOrder(Node08 node){
    if(node!=null){
        traversePostOrder(node.left);
        traversePostOrder(node.right);
        System.out.println(""+node.data);
    }
}

void traverseInOrder(Node08 node){
    if(node != null){
        traverseInOrder(node.left);
        System.out.println(""+node.data);
        traverseInOrder(node.right);
    }
}

Node08 getSuccessor(Node08 del){
    Node08 successor = del.right;
    Node08 successorParent = del;
    while(successor.left!=null){
        successorParent = successor;
```

```
        successor = successor.left;
    }
    if(successor!=del.right){
        successorParent.left = successor.right;
        successor.right = del.right;
    }
    return successor;
}

void delete(int data){
    if(isEmpty()){
        System.out.println("Tree is empty!");
        return;
    }
    Node08 parent = root;
    Node08 current = root;
    boolean isLeftChild = false;
    while(current!=null){
        if(current.data==data){
            break;
        }else if(data<current.data){
            parent = current;
            current = current.left;
            isLeftChild = true;
        }else if(data>current.data){
            parent = current;
            current = current.right;
            isLeftChild = false;
        }
    }
    if(current==null){
        System.out.println("Couldn't find data!");
        return;
    }else{
        if(current.left==null&&current.right==null){
            if(current==root){
                root=null;
            }else{
                if(isLeftChild){
                    parent.left=null;
                }else{
                    parent.right=null;
                }
            }
        }else if(current.left==null){
```

```
            if(current==root){
                root=current.right;
            }else{
                if(isLeftChild){
                    parent.left=current.right;
                }else{
                    parent.right=current.right;
                }
            }
        }else if(current.right==null){
            if(current==root){
                root=current.left;
            }else{
                if(isLeftChild){
                    parent.left=current.left;
                }else{
                    parent.right=current.left;
                }
            }
        }else{
            Node08 successor = getSuccessor(current);
            if(current==root){
                root=successor;
            }else{
                if(isLeftChild){
                    parent.left=successor;
                }else{
                    parent.right=successor;
                }
                successor.left=current.left;
            }
        }
    }
  }
}
```

```
package BinarySearchTree;

public class BinaryTreeArray08 {
  int[] data;
  int idxLast;

  public BinaryTreeArray08(){
    data = new int[10];
```

```
    }

    void populateData(int data[], int idxLast){
        this.data = data;
        this.idxLast = idxLast;
    }

    void traverseInOrder(int idxStart){
        if(idxStart<=idxLast){
            traverseInOrder((2*idxStart+1));
            System.out.println((data[idxStart]+" "));
            traverseInOrder(2*idxStart+2);
        }
    }
}
```

```
package BinarySearchTree;

public class BinaryTreeArrayMain08 {
    public static void main(String[] args) {
        BinaryTreeArray08 bta = new BinaryTreeArray08();
        int[] data = {6,4,8,3,5,7,9,0,0,0};
        int idxLast = 6;
        bta.populateData(data, idxLast);
        System.out.println("\n inOrder traversal:");
        bta.traverseInOrder(0);
        System.out.println("\n");
    }
}
```

```
package BinarySearchTree;

public class Node08 {
    int data;
    Node08 left;
    Node08 right;

    public Node08(){
    }
    public Node08(int data){
        this.left = null;
        this.data = data;
```

```
        this.right = null;
    }
}
```

```
package BinarySearchTree;

public class BinaryTreeMain08 {
    public static void main(String[] args) {
        BinaryTree08 bt = new BinaryTree08();
        bt.add(6);
        bt.add(4);
        bt.add(8);
        bt.add(3);
        bt.add(5);
        bt.add(7);
        bt.add(9);
        bt.add(10);
        bt.add(15);
        System.out.println("PreOrder Traversal:");
        bt.traversePreOrder(bt.root);
        System.out.println("");
        System.out.println("inOrder Traversal:");
        bt.traverseInOrder(bt.root);
        System.out.println("");
        System.out.println("PostOrder Traversal:");
        bt.traversePostOrder(bt.root);
        System.out.println("");
        System.out.println("Find node: " +bt.find(5));
        System.out.println("Delete node 8");
        bt.delete(8);
        System.err.println("");
        System.out.println("PreOrder Traversal:");
        bt.traversePreOrder(bt.root);
        System.out.println("");
    }
}
```

Question Lab unit 1

1. Why in a binary search tree, the process of searching data can be more effective than an ordinary binary tree?

Answer: Because the algorithm used is also quite easy, namely the search is carried out by comparing the word searched with the word at the root, if the first letter of the word searched is earlier than the root, the search will continue to the left, otherwise if it is later, the search will continue to the right.

2. What is the function of the left and right attributes in the Node class?

Answer: node direction

3. What is the Function of the root attribute in the BinaryTree class?

Answer: root is a node that has degree of exit >= 0 and degree of entry = 0.

4. What is the value of root when the tree object is first created??

Answer: null

5. What process will happen when the tree is empty, and a new node will be added?

Answer: If the tree is empty, then the new node is placed as the root of the tree.

6. Inside the add() method there is a program line as below. Explain what the program line is for?

Answer: to create a new node if it satisfies these conditions

Question Lab unit 2

1. What is the function of the data and idxLast attributes in the BinaryTreeArray class?

Answer: to get the largest value of the nodes contained in the binary search tree.

2. What is the function of the populateData() method?

Answer: to fill in the data

3. What is the function of the traverseInOrder() method?

Answer: makes a visit to the left subtree, prints the contents of the node that visited, then make a visit to the right subtree

4. If a binary tree node is stored in an array of index 2, then at what index are the left and right children at?

Answer: at index 1 and 3

5. What is the function of the statement int idxLast = 6 in lab-unit 2 experiment number 4?

Answer:

Assigment

1. Create a method in the BinaryTree class that will add nodes recursively.

Answer:

```
public boolean insertRecursiveBinaryTree (BinaryNode root, int pKey){
        boolean set=true;
        if(this.rootx==null) {
                BinaryNode b = new BinaryNode(pKey, null);
                this.rootx = b;
                System.out.println("set: "+ pKey);
                set = true;
        } else {
                if(pKey>root.key){
```

```
                    System.out.println("goto right leaf of: "+ root.key);
            if(root.right==null){
                    BinaryNode b = new BinaryNode(pKey, null);
                    root.right=b;
                    set = true;
                    System.out.println("current -> set "+b.key);
        } else {
                    return insertRecursiveBinaryTree(root.right,pKey);
        }
}
```

2. Create a method in the BinaryTree class to display the smallest and largest values in the tree.

Answer:

```
BinaryNode findMin (BinaryNode t) {
        if (t == null) throw exception;
        while (t.left != null) {
                t = t.left;
        }
        return t;
}
```

3. Create a method in the BinaryTree class to display the data in the leaf.

4. Create a method in the BinaryTree class to display how many leaves are in the tree.

Answer number 3 and 4 :

```
Public void getDaun(Node node){
        If ( node.nodeRight == null ){
                If (node.nodeLeft == null){
                        Leave++;
                } else {
                        getDaun(node.nodeLeft);
                }
        } else if ( node.nodeLeft == null){
                If (node.nodeKanan == null){
                        leave++;
                }else{
                        getDaun(node.nodeRight);
                } else if (node.nodeRight != null && node.nodeLeft != null){
                        getDaun(node.nodeLeft);
                        getDaun(node.nodeRight);
                }
}
```

5. Modify the BinaryTreeArray class , and add:

- add(int data) method to insert data into the tree
- traversePreOrder() and traversePostOrder() methods

Answer:

```
void push(struct tree* node,int a)
{
      if(root==NULL)
            root = newNode(a);
      else if(a < node->a && node->left==NULL)
      {
            node->left = newNode(a);
            node->left->parent = node;
      }
      else if(a > node->a && node->right==NULL)
      {
            node->right = newNode(a);
            node->right->parent = node;
      }
      else if(a < node->a)
            push(node->left,a);
      else
            push(node->right,a);
}
```

```
public void preorderTraversal() {
    preorder(root);
}
private void preorder(Node node){
    if(node == null) return;
    System.out.printf( "%d ", node.data );
    preorder(node.nodeKiri);
    preorder(node.nodeKanan);
}

public void postorderTraversal(){
    postorder( root );
}
private void postorder(Node node){
    if (node == null) return;
    postorder(node.nodeKiri);
    postorder(node.nodeKanan);
    System.out.printf( "%d ", node.data );
}
```