FLIP ROBO

WORKSHEET

FULL STACK DEVELOPMENT – WORKSHEET - 6

Ques 1. Write a java program that inserts a node into its proper sorted position in a sorted linked list.

Ans :

```
class LinkedList {
    Node head;


    class Node {
        int data;
        Node next;
        Node(int d)
        {
            data = d;
            next = null;
        }
    }


    void sortedInsert(Node new_node)
    {
        Node current;


        if (head == null || head.data
>= new_node.data) {
            new_node.next = head;
            head = new_node;
        }
        else {


            current = head;

            while (current.next != null
&& current.next.data < new_node.data) {

              current = current.next;
            }

            new_node.next = current.next;
            current.next = new_node;
        }
    }

    /*Utility functions*/
    Node newNode(int data)
    {
        Node x = new Node(data);
        return x;
    }


    void printList()
    {
        Node temp = head;
        while (temp != null) {
            System.out.print(temp.data + " ");
            temp = temp.next;
        }
    }

        public static void main(String args[])
    {
        LinkedList llist = new LinkedList();
        Node new_node;
        new_node = llist.newNode(5);
        llist.sortedInsert(new_node);
        new_node = llist.newNode(10);
        llist.sortedInsert(new_node);
        new_node = llist.newNode(7);
        llist.sortedInsert(new_node);
        new_node = llist.newNode(3);
        llist.sortedInsert(new_node);
        new_node = llist.newNode(1);
        llist.sortedInsert(new_node);
        new_node = llist.newNode(9);
        llist.sortedInsert(new_node);
        System.out.println("Created Linked List");
        llist.printList();
    }
}
```

Ques 2. Write a java program to compute the height of the binary tree.

Ans :

```
class Node {
    int data;
    Node left, right;

    Node(int item)
    {
        data = item;
        left = right = null;
    }
}

class BinaryTree {
    Node root;
    int maxDepth(Node node)
    {
        if (node == null)
            return 0;
        else {

            int lDepth = maxDepth(node.left);
            int rDepth = maxDepth(node.right);


            if (lDepth > rDepth)
                return (lDepth + 1);
            else
                return (rDepth + 1);
        }
    }


    public static void main(String[] args)
    {
        BinaryTree tree = new BinaryTree();

        tree.root = new Node(2);
        tree.root.left = new Node(5);
        tree.root.right = new Node(7);
        tree.root.left.left = new Node(9);
        tree.root.left.right = new Node(10);

        System.out.println("Height of tree is "
                           + tree.maxDepth(tree.root));
    }
}
```

Ques 3. Write a java program to determine whether a given binary tree is a BST or not.

Ans :

```
import java.io.*;

class GFG {

  static class node {
    int data;
    node left, right;
  }
```

```
static node newNode(int data)
{
  node Node = new node();
  Node.data = data;
  Node.left = Node.right = null;

  return Node;
}

static int maxValue(node Node)
{
  if (Node == null) {
    return Integer.MIN_VALUE;
  }
  int value = Node.data;
  int leftMax = maxValue(Node.left);
  int rightMax = maxValue(Node.right);

  return Math.max(value, Math.max(leftMax, rightMax));
}

static int minValue(node Node)
{
  if (Node == null) {
    return Integer.MAX_VALUE;
  }
  int value = Node.data;
  int leftMax = minValue(Node.left);
  int rightMax = minValue(Node.right);

  return Math.min(value, Math.min(leftMax, rightMax));
}


static int isBST(node Node)
{
  if (Node == null) {
    return 1;
  }


  if (Node.left != null
      && maxValue(Node.left) > Node.data) {
    return 0;
  }


  if (Node.right != null
      && minValue(Node.right) < Node.data) {
    return 0;
  }
```

```
    if (isBST(Node.left) != 1
        || isBST(Node.right) != 1) {
      return 0;
    }


    return 1;
  }

  public static void main(String[] args)
  {
    node root = newNode(4);
    root.left = newNode(2);
    root.right = newNode(5);

    root.left.left = newNode(1);
    root.left.right = newNode(3);

        if (isBST(root) == 1) {
      System.out.print("Is BST");
    }
    else {
      System.out.print("Not a BST");
    }
  }
}
```

Ques 4. Write a java code to Check the given below expression is balanced or not . (using stack)

{ { [[(( ))]]) } }

```
import java.util.*;

public class BalancedBrackets {


    static boolean areBracketsBalanced(String expr)
    {

        Deque<Character> stack
            = new ArrayDeque<Character>();


        for (int i = 0; i < expr.length(); i++) {
            char x = expr.charAt(i);

            if (x == '(' || x == '[' || x == '{') {

                stack.push(x);
                continue;
            }
            if (stack.isEmpty())
                return false;
            char check;
            switch (x) {
            case ')':
                check = stack.pop();
                if (check == '{' || check == '[')
                    return false;
                break;

            case '}':
                check = stack.pop();
                if (check == '(' || check == '[')
                    return false;
                break;

            case ']':
                check = stack.pop();
                if (check == '(' || check == '{')
                    return false;
                break;
            }
        }


        return (stack.isEmpty());
    }


    public static void main(String[] args)
    {
        String expr = "([{}])";


        if (areBracketsBalanced(expr))
            System.out.println("Balanced ");
        else
            System.out.println("Not Balanced ");
    }
}
```

Ques 5. Write a java program to Print left view of a binary tree using queue.

Ans :

```
import java.util.*;

class GFG {
        static class Node {
        int data;
        Node left, right;
        public Node(int item)
        {
            data = item;
            left = right = null;
        }
    };

    public static ArrayList<Integer> leftView(Node root)
    {

        ArrayList<Integer> ans = new ArrayList<>();

        if (root == null) {
            return ans;
        }

        Queue<Node> q = new LinkedList<>();
        q.add(root);
        q.add(null);
        boolean ok = true;

        while (!q.isEmpty()) {
            Node it = q.poll();
            if (it == null) {
                if (ok == false) {
                    ok = true;
                }

                if (q.size() == 0)
                    break;

                else {
                    q.add(null);
                }
            }
            else {

                if (ok) {
                    ans.add(it.data);
                    ok = false;
                }

                if (it.left != null) {
                    q.add(it.left);
                }

                if (it.right != null) {
                    q.add(it.right);
                }
            }
        }

        return ans;
    }
    public static void main(String[] args)
    {
        Node root = new Node(2);
        root.left = new Node(7);
        root.right = new Node(9);
        root.left.left = new Node(13);
        root.left.right = new Node(55);
        root.right.right = new Node(133);
        root.right.left = new Node(10);
        root.right.right.left = new Node(14);

        ArrayList<Integer> vec = leftView(root);
        for (int x : vec) {
            System.out.print(x + " ");
        }
        System.out.println();
    }
}
```