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.

```
package com.java.LinkedList;

class Node {
	int data;
	Node next;

	public Node(int data) {
		this.data=data;
		this.next=null;
	}
}

class LinkedList1 {
	Node head;
	public LinkedList1() {
		head=null;
		}
	public void insertSorted(int data) {
		Node newNode= new Node(data);
		// if the lists is empty of the new node data is smaller than the
			current head data,
		// make new node the head and point it to the old head.
		if(head==null||data<head.data) {
			newNode.next=head;
			head=newNode;
		}
		else {
			// find teh porpoer position to insert the new node while
				maintaining the insertion order.
			Node current=head;
			while(current.next!=null&&current.next.data<data) {
				current=current.next;
			}
			newNode.next=current.next;
			current.next=newNode;
		}
	}

	public void display() {
		Node current =head;
		while(current!=null) {
			System.out.print(current.data+" ");
			current=current.next;
		}
		System.out.println("Nulls");
	}
}
```

```
public class SortedLinkedList_123 {
      public static void main(String[] args) {
            LinkedList1 linkedlist = new LinkedList1();

            //Add elements one by one
            linkedlist.insertSorted(10);
            linkedlist.insertSorted(5);
            linkedlist.insertSorted(20);
            linkedlist.insertSorted(15);
            linkedlist.insertSorted(25);

            // Display the sorted list
            linkedlist.display();
      }

}
```

Explanation: In this program, we have a Node class to represent individual nodes in the linked list. The SortedLinkedList class has a head pointer to the first node of the linked list. The insertSorted() method inserts a new node with the given data in its proper sorted position in the linked list. The main() method creates a sorted linked list and inserts elements in sorted order. Finally, it displays the sorted linked list.

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

Ans:

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

   public Node(int data) {
      this.data = data;
      this.left = null;
      this.right = null;
   }
}

public class BinaryTreeHeight {
   Node root;

   public int height(Node node) {
      if (node == null) {
         return 0;
      } else {
         int leftHeight = height(node.left);
         int rightHeight = height(node.right);

         return Math.max(leftHeight, rightHeight) + 1;
      }
   }

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

      // Construct the binary tree
      tree.root = new Node(1);
      tree.root.left = new Node(2);
```

```
        tree.root.right = new Node(3);
        tree.root.left.left = new Node(4);
        tree.root.left.right = new Node(5);
        tree.root.right.left = new Node(6);
        tree.root.right.right = new Node(7);

        int height = tree.height(tree.root);
        System.out.println("Height of the binary tree is: " + height);
    }
}
```

Explanation: In this program, we have a Node class to represent individual nodes in the binary tree. The BinaryTreeHeight class has a height() method that calculates the height of the binary tree recursively. The main() method creates a binary tree and calls the height() method to find the height of the tree. The height of the binary tree is the length of the longest path from the root to any leaf node.

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

```java
package com.java.SixthAssignement;

class TreeNode {
    int val;
    TreeNode left;
    TreeNode right;

    public TreeNode(int val) {
        this.val = val;
    }
}

public class BinaryTreeChecker {
    // Variable to keep track of the previous node during in-order traversal
    TreeNode prev;

    // Function to check whether the binary tree is a binary search tree or not
    public boolean isBST(TreeNode root) {
        prev = null; // Initialize the prev variable

        // Call the helper function to perform in-order traversal
        return isBSTHelper(root);
    }

    // Helper function to perform in-order traversal and check for sorted order
    private boolean isBSTHelper(TreeNode node) {
        // Base case: If the node is null, it is considered a valid BST node
        if (node == null) {
            return true;
        }

        // Check the left subtree
        if (!isBSTHelper(node.left)) {
            return false;
        }

        // Check the current node's value against the previous node's value
```

```
        if (prev != null && node.val <= prev.val) {
            return false;
        }

        // Update the previous node to the current node
        prev = node;

        // Check the right subtree
        return isBSTHelper(node.right);
    }

    public static void main(String[] args) {
        // Example usage
        TreeNode root = new TreeNode(4);
        root.left = new TreeNode(2);
        root.right = new TreeNode(6);
        root.left.left = new TreeNode(1);
        root.left.right = new TreeNode(3);

        BinaryTreeChecker checker = new BinaryTreeChecker();
        boolean isBST = checker.isBST(root);
        System.out.println("Is the binary tree a binary search tree? " +
isBST);
    }
}
```

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

```
import java.util.Stack;

public class BalancedExpressionChecker {

    public static boolean isBalanced(String expression) {

        Stack<Character> stack = new Stack<>();

        for (int i = 0; i < expression.length(); i++) {

            char ch = expression.charAt(i);

            if (ch == '(' || ch == '[' || ch == '{') {
                stack.push(ch);
            } else if (ch == ')' || ch == ']' || ch == '}') {
                if (stack.isEmpty()) {
                    return false;
                }

                char top = stack.pop();

                if ((ch == ')' && top != '(') || (ch == ']' && top != '[') || (ch == '}' && top != '{'))
{
                    return false;
                }
            }
        }

        return stack.isEmpty();
    }
```

```
    public static void main(String[] args) {
        String expression = "{{[[(())]]}}";
        if (isBalanced(expression)) {
            System.out.println("The expression is balanced.");
        } else {
            System.out.println("The expression is not balanced.");
        }
    }
}
```

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

```
package com.java.SixthAssignement;

import java.util.LinkedList;
import java.util.Queue;

class Node {
    int data;
    Node left, right;

    public Node(int data) {
        this.data = data;
        this.left = this.right = null;
    }
}

public class LeftViewBinaryTreeUsingQueue {
    public static void printLeftView(Node root) {
        if (root == null) {
            return;
        }

        Queue<Node> queue = new LinkedList<>();
        queue.add(root);

        while (!queue.isEmpty()) {
            int size = queue.size();
            boolean isFirstNode = true;

            for (int i = 0; i < size; i++) {
                Node node = queue.poll();

                if (isFirstNode) {
                    System.out.print(node.data + " ");
                    isFirstNode = false;
                }

                if (node.left != null) {
                    queue.add(node.left);
                }
```

```
                if (node.right != null) {
                    queue.add(node.right);
                }
            }
        }
    }

    public static void main(String[] args) {
        Node root = new Node(1);
        root.left = new Node(2);
        root.right = new Node(3);
        root.left.left = new Node(4);
        root.left.right = new Node(5);
        root.right.left = new Node(6);
        root.right.right = new Node(7);

        System.out.print("Left View of Binary Tree: ");
        printLeftView(root);
    }
}
```

Explanation: **In this program, we use a queue to traverse the binary tree level by level. While traversing each level, we print the first node encountered (i.e., the leftmost node) at that level. This gives us the left view of the binary tree. The printLeftView() method takes the root node of the binary tree as input and prints the left view of the tree. In the main() method, we create a binary tree and call the printLeftView() method to print its left view.**