|  |  |  |
| --- | --- | --- |
| **Academic Year:** 2024-25 | **Year:** Third Year | **Term:** II |
| **PRN No.: 1012412079** | **Name: Ratnajeet Patil** | |
| **Subject:** Python Programming | | |
| **Assignment No.**: 1 |  | |
| **Date:** |  | |

**Lab Assignment: 01**

**Title:** Create a binary tree and perform inorder , preorder and postorder traversals.

1. import java.util.Scanner;

2. import java.util.Stack;

3.

4. class Node {

5.     int data;

6.     Node left, right;

7.

8.     public Node(int data) {

9.         this.data = data;

10.         left = right = null;

11.     }

12. }

13.

14. class BinaryTree {

15.     Node root;

16.     Scanner scanner = new Scanner(System.in);

17.

18.     public void insert(int data) {

19.         if (root == null) {

20.             root = new Node(data);

21.             System.out.println("Inserted as root node.");

22.             return;

23.         }

24.

25.         Node current = root;

26.         while (true) {

27.             System.out.print("Enter direction (L/R) from " + current.data + ": ");

28.             String direction = scanner.next();

29.             if (direction.equals("L")) {

30.                 if (current.left == null) {

31.                     current.left = new Node(data);

32.                     System.out.println("Inserted at left of " + current.data);

33.                     break;

34.                 } else {

35.                     current = current.left;

36.                 }

37.             } else if (direction.equals("R")) {

38.                 if (current.right == null) {

39.                     current.right = new Node(data);

40.                     System.out.println("Inserted at right of " + current.data);

41.                     break;

42.                 } else {

43.                     current = current.right;

44.                 }

45.             } else {

46.                 System.out.println("Invalid direction! Enter either 'left' or 'right'.");

47.             }

48.         }

49.     }

50.

51.     public void inOrderTraversal() {

52.         if (root == null)

53.             return;

54.

55.         Stack<Node> stack = new Stack<>();

56.         Node current = root;

57.

58.         while (current != null || !stack.isEmpty()) {

59.             // Reach leftmost node of current node

60.             while (current != null) {

61.                 stack.push(current);

62.                 current = current.left;

63.             }

64.

65.             // Current is null at this point

66.             current = stack.pop();

67.             System.out.print(current.data + " ");

68.

69.             // Move to the right subtree

70.             current = current.right;

71.         }

72.     }

73.

74.     public void preOrderTraversal() {

75.         if (root == null)

76.             return;

77.

78.         Stack<Node> stack = new Stack<>();

79.         stack.push(root);

80.

81.         while (!stack.isEmpty()) {

82.             // Pop the top node and print it

83.             Node current = stack.pop();

84.             System.out.print(current.data + " ");

85.

86.             // Push right child first so that left child is processed first

87.             if (current.right != null) {

88.                 stack.push(current.right);

89.             }

90.             if (current.left != null) {

91.                 stack.push(current.left);

92.             }

93.         }

94.     }

95.

96.     public void postOrderTraversal() {

97.         if (root == null)

98.             return;

99.

100.         Stack<Node> stack1 = new Stack<>();

101.         Stack<Node> stack2 = new Stack<>();

102.

103.         stack1.push(root);

104.

105.         // First stack to get nodes in reverse post-order

106.         while (!stack1.isEmpty()) {

107.             Node current = stack1.pop();

108.             stack2.push(current);

109.

110.             if (current.left != null) {

111.                 stack1.push(current.left);

112.             }

113.             if (current.right != null) {

114.                 stack1.push(current.right);

115.             }

116.         }

117.

118.         // Second stack has nodes in post-order

119.         while (!stack2.isEmpty()) {

120.             Node current = stack2.pop();

121.             System.out.print(current.data + " ");

122.         }

123.     }

124.

125.     // Public methods to call the traversal methods

126.     public void inOrder() {

127.         System.out.println("In-order Traversal of the Tree (Iterative):");

128.         inOrderTraversal();

129.         System.out.println();

130.     }

131.

132.     public void preOrder() {

133.         System.out.println("Pre-order Traversal of the Tree (Iterative):");

134.         preOrderTraversal();

135.         System.out.println();

136.     }

137.

138.     public void postOrder() {

139.         System.out.println("Post-order Traversal of the Tree (Iterative):");

140.         postOrderTraversal();

141.         System.out.println();

142.     }

143. }

144.

145. public class BinaryTreeUserInput {

146.     public static void main(String[] args) {

147.         Scanner scanner = new Scanner(System.in);

148.         BinaryTree tree = new BinaryTree();

149.         int choice;

150.         boolean exit = false;

151.

152.         System.out.print("Enter number of nodes for the initial tree: ");

153.         int n = scanner.nextInt();

154.

155.         for (int i = 0; i < n; i++) {

156.             System.out.print("Enter value for node: ");

157.             int value = scanner.nextInt();

158.             tree.insert(value);

159.         }

160.

161.         while (!exit) {

162.             System.out.println("\nBinary Tree Operations:");

163.             System.out.println("1. Insert a node");

164.             System.out.println("2. Pre-order traversal");

165.             System.out.println("3. In-order traversal");

166.             System.out.println("4. Post-order traversal");

167.             System.out.println("5. Exit");

168.             System.out.print("Enter your choice: ");

169.

170.             choice = scanner.nextInt();

171.

172.             switch (choice) {

173.                 case 1:

174.                     System.out.print("Enter value for new node: ");

175.                     int value = scanner.nextInt();

176.                     tree.insert(value);

177.                     break;

178.                 case 2:

179.                     tree.preOrder();

180.                     break;

181.                 case 3:

182.                     tree.inOrder();

183.                     break;

184.                 case 4:

185.                     tree.postOrder();

186.                     break;

187.                 case 5:

188.                     exit = true;

189.                     System.out.println("Exiting program...");

190.                     break;

191.                 default:

192.                     System.out.println("Invalid choice! Please try again.");

193.             }

194.         }

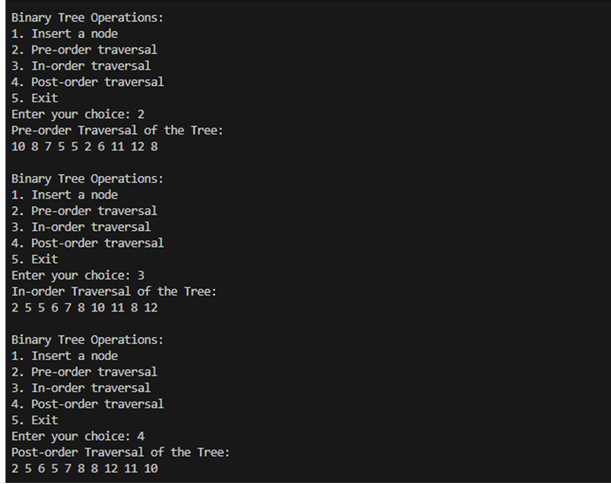
195.         scanner.close();

196.     }

197. }

198.

199.

****