// non-recursive java program for inorder traversal

/\* importing the necessary class \*/

import java.util.Stack;

/\* Class containing left and right child of current

node and key value\*/

class Node {

int data;

Node left, right;

public Node(int item) {

data = item;

left = right = null;

}

}

/\* Class to print the inorder traversal \*/

class BinaryTree {

Node root;

void inorder() {

if (root == null) {

return;

}

//keep the nodes in the path that are waiting to be visited

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

Node node = root;

//first node to be visited will be the left one

while (node != null) {

stack.push(node);

node = node.left;

}

// traverse the tree

while (stack.size() > 0) {

// visit the top node

node = stack.pop();

System.out.print(node.data + " ");

if (node.right != null) {

node = node.right;

// the next node to be visited is the leftmost

while (node != null) {

stack.push(node);

node = node.left;

}

}

}

}

|  |
| --- |
| //////////////////=POSTORDER=//////////////////////////////  class BinaryTree      static Node root;      ArrayList<Integer> list = new ArrayList<Integer>();      // An iterative function to do postorder traversal of a given binary tree      ArrayList<Integer> postOrderIterative(Node node) {          Stack<Node> S = new Stack<Node>();          // Check for empty tree          if (node == null) {              return list;          }          S.push(node);          Node prev = null;          while (!S.isEmpty()) {              Node current = S.peek();              /\* go down the tree in search of a leaf an if soprocessitand pop              stack otherwise move down \*/              if (prev == null || prev.left == current || prev.right == current) {                  if (current.left != null) {                      S.push(current.left);                  } else if (current.right != null) {                      S.push(current.right);                  } else {                      S.pop();                      list.add(current.data);                  }                    /\* go up the tree from left node, if the child is right                  push it onto stack otherwise process parent and pop stack \*/              } else if (current.left == prev) {                  if (current.right != null) {                      S.push(current.right);                  } else {                      S.pop();                      list.add(current.data);                  }                    /\* go up the tree from right node and after coming back                   from right node process parent and pop stack \*/              } else if (current.right == prev) {                  S.pop();                  list.add(current.data);              }                prev = current;          }            return list;      } |

///////////////-PREORDER-/////////////////////////

    void iterativePreorder(){

        iterativePreorder(root);

    }

    // An iterative process to print preorder traversal of Binary tree

    void iterativePreorder(Node node) {

        // Base Case

        if (node == null) {

            return;

        }

        // Create an empty stack and push root to it

        Stack<Node> nodeStack = new Stack<Node>();

        nodeStack.push(root);

        /\* Pop all items one by one. Do following for every popped item

         a) print it

         b) push its right child

         c) push its left child

         Note that right child is pushed first so that left is processed first \*/

        while (nodeStack.empty() == false) {

            // Pop the top item from stack and print it

            Node mynode = nodeStack.peek();

            System.out.print(mynode.data + " ");

            nodeStack.pop();

            // Push right and left children of the popped node to stack

            if (mynode.right != null) {

                nodeStack.push(mynode.right);

            }

            if (mynode.left != null) {

                nodeStack.push(mynode.left);

            }

        }

    }

public static void main(String args[]) {

/\* creating a binary tree and entering

the nodes \*/

BinaryTree tree = new BinaryTree();

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.inorder();

tree.postorder();

tree.preorder();

}

}