We have discussed a simple [iterative postorder traversal using two stacks](https://www.geeksforgeeks.org/iterative-postorder-traversal/) in the previous post. In this post, an approach with only one stack is discussed.

The idea is to move down to leftmost node using left pointer. While moving down, push root and root’s right child to stack. Once we reach leftmost node, print it if it doesn’t have a right child. If it has a right child, then change root so that the right child is processed before.

Following is detailed algorithm.

1.1 Create an empty stack

2.1 Do following while root is not NULL

a) Push root's right child and then root to stack.

b) Set root as root's left child.

2.2 Pop an item from stack and set it as root.

a) If the popped item has a right child and the right child

is at top of stack, then remove the right child from stack,

push the root back and set root as root's right child.

b) Else print root's data and set root as NULL.

2.3 Repeat steps 2.1 and 2.2 while stack is not empty.

Let us consider the following tree 

Diagram

Description automatically generated

Following are the steps to print postorder traversal of the above tree using one stack.

1. Right child of 1 exists.

Push 3 to stack. Push 1 to stack. Move to left child.

Stack: 3, 1

2. Right child of 2 exists.

Push 5 to stack. Push 2 to stack. Move to left child.

Stack: 3, 1, 5, 2

3. Right child of 4 doesn't exist. '

Push 4 to stack. Move to left child.

Stack: 3, 1, 5, 2, 4

4. Current node is NULL.

Pop 4 from stack. Right child of 4 doesn't exist.

Print 4. Set current node to NULL.

Stack: 3, 1, 5, 2

5. Current node is NULL.

Pop 2 from stack. Since right child of 2 equals stack top element,

pop 5 from stack. Now push 2 to stack.

Move current node to right child of 2 i.e. 5

Stack: 3, 1, 2

6. Right child of 5 doesn't exist. Push 5 to stack. Move to left child.

Stack: 3, 1, 2, 5

7. Current node is NULL. Pop 5 from stack. Right child of 5 doesn't exist.

Print 5. Set current node to NULL.

Stack: 3, 1, 2

8. Current node is NULL. Pop 2 from stack.

Right child of 2 is not equal to stack top element.

Print 2. Set current node to NULL.

Stack: 3, 1

9. Current node is NULL. Pop 1 from stack.

Since right child of 1 equals stack top element, pop 3 from stack.

Now push 1 to stack. Move current node to right child of 1 i.e. 3

Stack: 1

10. Repeat the same as above steps and Print 6, 7 and 3.

Pop 1 and Print 1.

[Recommended: Please try your approach on ***{IDE}*** first, before moving on to the solution.](https://ide.geeksforgeeks.org/)

* C
* Java
* Python3
* C#
* Javascript

|  |
| --- |
| // A java program for iterative postorder traversal using stack    **import** java.util.ArrayList;  **import** java.util.Stack;    // A binary tree node  **class** Node  {  **int** data;      Node left, right;        Node(**int** item)      {          data = item;          left = right;      }  }    **class** BinaryTree  {      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 so process it              and 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;      }        // Driver program to test above functions  **public** **static** **void** main(String args[])      {      BinaryTree tree = **new** BinaryTree();            // Let us create trees shown in above diagram          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);            ArrayList<Integer> mylist = tree.postOrderIterative(tree.root);            System.out.println("Post order traversal of binary tree is :");          System.out.println(mylist);      }  }    // This code has been contributed by Mayank Jaiswal |

**Output**

Post order traversal of binary tree is :

[4 5 2 6 7 3 1 ]

**Method 2:**   
Push directly root node two times while traversing to the left. While popping if you find stack top() is same as root then go for root->right else print root.

* Java
* Python3
* C#
* Javascript

|  |
| --- |
| // Simple Java program to print PostOrder Traversal(Iterative)  **import** java.util.Stack;    // A binary tree node  **class** Node  {  **int** data;      Node left, right;        Node(**int** item)      {          data = item;          left = right;      }  }    // create a postorder class  **class** PostOrder  {      Node root;        // An iterative function to do postorder traversal      // of a given binary tree  **private** **void** postOrderIterative(Node root) {          Stack<Node> stack = **new** Stack<>();  **while**(**true**) {  **while**(root != **null**) {                  stack.push(root);                  stack.push(root);                  root = root.left;              }                // Check for empty stack  **if**(stack.empty()) **return**;              root = stack.pop();    **if**(!stack.empty() && stack.peek() == root) root = root.right;    **else** {                    System.out.print(root.data + " "); root = **null**;              }          }      }        // Driver program to test above functions  **public** **static** **void** main(String args[])      {      PostOrder tree = **new** PostOrder();            // Let us create trees shown in above diagram          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);          System.out.println("Post order traversal of binary tree is :");          tree.postOrderIterative(tree.root);      }  } |

Post order traversal of binary tree is :

4 5 2 6 7 3 1

**Output**

Post order traversal of binary tree is :

4 5 2 6 7 3 1