**Check for Symmetric Binary Tree (Iterative Approach)**

**Java:**

We have discussed recursive approach to solve this problem in below post :  
[Symmetric Tree (Mirror Image of itself)](https://www.geeksforgeeks.org/symmetric-tree-tree-which-is-mirror-image-of-itself/)  
In this post, iterative approach is discussed. We use Queue here. Note that for a symmetric tree elements at every level are palindromic. In example 2, at the leaf level, the elements are not palindromic.   
In other words,

The left child of left subtree = right child of right subtree.

The right child of left subtree = left child of right subtree.

If we insert the left child of left subtree first followed by right child of the right subtree in the queue, we only need to ensure that these are equal.   
Similarly, If we insert the right child of left subtree followed by left child of the right subtree in the queue, we again need to ensure that these are equal.

Below is the implementation based on above idea.

C++

Java

Python3

C#

Javascript

|  |
| --- |
| // Iterative Java program to check if  // given binary tree symmetric  import java.util.\* ;    public class BinaryTree  {      Node root;      static class Node      {          int val;          Node left, right;          Node(int v)          {              val = v;              left = null;              right = null;          }      }        /\* constructor to initialise the root \*/      BinaryTree(Node r) { root = r; }        /\* empty constructor \*/      BinaryTree()  {  }          /\* function to check if the tree is Symmetric \*/      public boolean isSymmetric(Node root)      {          /\* This allows adding null elements to the queue \*/          Queue<Node> q = new LinkedList<Node>();            /\* Initially, add left and right nodes of root \*/          q.add(root.left);          q.add(root.right);            while (!q.isEmpty())          {              /\* remove the front 2 nodes to                check for equality \*/              Node tempLeft = q.remove();              Node tempRight = q.remove();                /\* if both are null, continue and check                 for further elements \*/              if (tempLeft==null && tempRight==null)                  continue;                /\* if only one is null---inequality, return false \*/              if ((tempLeft==null && tempRight!=null) ||                  (tempLeft!=null && tempRight==null))                  return false;                /\* if both left and right nodes exist, but                 have different values-- inequality,                 return false\*/              if (tempLeft.val != tempRight.val)                  return false;                /\* Note the order of insertion of elements                 to the queue :                 1) left child of left subtree                 2) right child of right subtree                 3) right child of left subtree                 4) left child of right subtree \*/              q.add(tempLeft.left);              q.add(tempRight.right);              q.add(tempLeft.right);              q.add(tempRight.left);          }            /\* if the flow reaches here, return true\*/          return true;      }        /\* driver function to test other functions \*/      public static void main(String[] args)      {          Node n = new Node(1);          BinaryTree bt = new BinaryTree(n);          bt.root.left = new Node(2);          bt.root.right = new Node(2);          bt.root.left.left = new Node(3);          bt.root.left.right = new Node(4);          bt.root.right.left = new Node(4);          bt.root.right.right = new Node(3);            if (bt.isSymmetric(bt.root))              System.out.println("The given tree is Symmetric");          else              System.out.println("The given tree is not Symmetric");      }  } |