Feb2025 ADS Exam

Given the head of a singly linked list, reverse the list and return the new head.

public static SinglyLinkedListNode reverse(SinglyLinkedListNode head)
Note:

Do not use collection Framework
Input Format

The first line contains an integer n, the number of elements in the linked list.
The next n lines each contain an integer, the elements of the linked list.
Output Format

Print the reversed linked list in a single line, space-separated.
Sample Input 0

```
5
1
2
3
4
5
```

Sample Output 0

```
5 4 3 2 1
```

```
import java.io.*;
import java.util.*;

class SinglyLinkedListNode {
   int data ;
   SinglyLinkedListNode next ;

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

public class Solution {

      /* Enter your code here. Read input from STDIN. Print output to STDOUT. Your class should be named Solution.
*/
      public static SinglyLinkedListNode reverse (SinglyLinkedListNode head) {
         SinglyLinkedListNode prev = null ;
         SinglyLinkedListNode current = head ;
         SinglyLinkedListNode next = null ;

         while (current != null) {
            next = current.next ;
            current.next = prev ;
            prev = current ;
            current = next ;
         }
         return prev ;
      }
      static void printList (SinglyLinkedListNode head) {
         SinglyLinkedListNode current = head ;
         while (current != null) {
            System.out.print (current.data + " ");
            current = current.next ;
         }
      }

      public static void main(String[] args) {

         Scanner sc = new Scanner (System.in) ;
         int n = sc.nextInt ();

         SinglyLinkedListNode head = null, tail = null ;
         for (int i = 0; i < n; i++) {
            int data = sc.nextInt ();

            SinglyLinkedListNode newNode = new SinglyLinkedListNode (data) ;
            if (head == null) {
               head = newNode ;
               tail = head ;
            }
            else {
               tail.next = newNode;
               tail = newNode ;
            }
         }
         SinglyLinkedListNode reversedHead = reverse (head);
         printList (reversedHead) ;

   }
}
```

===

A sentence is a sequence of characters. Write a recursive Java program that reverses the entire string. That is, the character at index [0] becomes the last, the character at index [1] becomes the second last, and so on.

Function Signature:

public static String reverseString(String str)
Note:

Do not use collection Framework
Input Format

A single line string S (no spaces, alphanumeric).

Output Format

A single line: the reversed string.

Sample Input 0
CDACMumbai

Sample Output 0
iabmuMCADC

```
import java.io.*;
import java.util.*;


public class Soluction {

  public static String reverseString(String str) {
      if (str.length() == 0)
         return "";
      return reverseString(str.substring(1)) + str.charAt(0);
   }

   public static void main (String[] args) {
      Scanner sc = new Scanner(System.in) ;
      String s = sc.nextLine();
      System.out.println (reverseString(s));
   }
}


/*

public class Solution {

   static void reverse (char[] str, int index) {
      if (index < 0) {
         return ;
      }
      System.out.println (str [index]);
      reverse (str, index-1) ;
   }

   public static void main(String[] args) {
      Scanner sc = new Scanner (System.in) ;
      String input = sc.nextLine ();
      char[] strArray = new char[input.length()];
      for (int i = 0; i <= input.lenght(); i++) {
         strArray [i] = input.charAt(i) ;
      }
      System.out.println ("Reversed :");
      reverse(strArray, strArray.length -1);
      /* Enter your code here. Read input from STDIN. Print output to STDOUT. Your class should be named Solution.
*/

  //  }
// }

//*/

=============================================================================
```

You are given a series of integers to insert into a Binary Search Tree (BST). After all insertions, delete a given node from the BST and print the in-order traversal of the updated tree.

Function Signatures:

public static Node insert(Node root, int data)

public static Node delete(Node root, int data)

public static void traverse(Node root)
Note:

Do not use collection Framework
Input Format

The first line contains an integer n — the number of nodes to insert.
The next n lines contain one integer each — the data values to insert.
The last line contains an integer — the node value to delete.
Output Format

Print the in-order traversal of the BST after the deletion.
Sample Input 0

```
6
50
30
20
40
70
60
50
```

Sample Output 0

```
20 30 40 60 70
```

```
import java.util.Scanner;

public class Solution {

    static class Node {
        int data;
        Node left, right;

        Node(int value) {
            data = value;
            left = right = null;
        }
    }

    // Insert node into BST
    public static Node insert(Node root, int data) {
        if (root == null)
            return new Node(data);
        if (data < root.data)
            root.left = insert(root.left, data);
        else if (data > root.data)
            root.right = insert(root.right, data);
        return root;
    }

    // Delete node from BST
    public static Node delete(Node root, int key) {
        if (root == null)
            return null;

        if (key < root.data)
            root.left = delete(root.left, key);
        else if (key > root.data)
            root.right = delete(root.right, key);
        else {
            if (root.left == null)
                return root.right;
            else if (root.right == null)
                return root.left;

            root.data = minValue(root.right);
            root.right = delete(root.right, root.data);
        }
        return root;
    }

    public static int minValue(Node root) {
        while (root.left != null)
            root = root.left;
        return root.data;
    }

    // In-order traversal
    public static void inorder(Node root) {
        if (root != null) {
            inorder(root.left);
            System.out.print(root.data + " ");
            inorder(root.right);
        }
    }

    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);

        int n = sc.nextInt();
        Node root = null;

        for (int i = 0; i < n; i++) {
            int val = sc.nextInt();
            root = insert(root, val);
        }

        int deleteVal = sc.nextInt();
        root = delete(root, deleteVal);

        inorder(root);
    }
}
```