//Activity R-8.5

//getLeafCount(node)

//1) If node is NULL then return 0.

//2) Else If left and right child nodes are NULL return 1.

//3) Else recursively calculate leaf count of the tree using below formula.

Leaf count of a tree = Leaf count of left subtree +

Leaf count of right subtree

package Tree;

//Java implementation to find leaf count of a given Binary tree

/\* 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;

}

}

public class BinaryTree

{

//Root of the Binary Tree

Node root;

/\* Function to get the count of leaf nodes in a binary tree\*/

int getLeafCount()

{

return getLeafCount(root);

}

int getLeafCount(Node node)

{

if (node == null)

return 0;

if (node.left == null && node.right == null)

return 1;

else

return getLeafCount(node.left) + getLeafCount(node.right);

}

/\* Driver program to test above functions \*/

public static void main(String args[])

{

/\* create a tree \*/

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);

/\* get leaf count of the above tree \*/

System.out.println("The leaf count of binary tree is : "

+ tree.getLeafCount());

}

}

//End of Code

package NewActivity;

//Activity R-6.4

//Activity C-6.17

//enQueue(q, x)

// 1) Push x to stack1 (assuming size of stacks is unlimited).

//Here time complexity will be O(1)

//deQueue(q)

// 1) If both stacks are empty then error.

// 2) If stack2 is empty

// While stack1 is not empty, push everything from stack1 to stack2.

// 3) Pop the element from stack2 and return it.

//Here time complexity will be O(n)

/\* Java Program to implement a queue using two stacks \*/

// Note that Stack class is used for Stack implementation

import java.util.Stack;

public class New {

/\* class of queue having two stacks \*/

static class Queue {

Stack<Integer> stack1;

Stack<Integer> stack2;

}

/\* Function to push an item to stack\*/

static void push(Stack<Integer> top\_ref, int new\_data)

{

// Push the data onto the stack

top\_ref.push(new\_data);

}

/\* Function to pop an item from stack\*/

static int pop(Stack<Integer> top\_ref)

{

/\*If stack is empty then error \*/

if (top\_ref.isEmpty()) {

System.out.println("Stack Underflow");

System.exit(0);

}

// pop the data from the stack

return top\_ref.pop();

}

// Function to enqueue an item to the queue

static void enQueue(Queue q, int x)

{

push(q.stack1, x);

}

/\* Function to deQueue an item from queue \*/

static int deQueue(Queue q)

{

int x;

/\* If both stacks are empty then error \*/

if (q.stack1.isEmpty() && q.stack2.isEmpty()) {

System.out.println("Q is empty");

System.exit(0);

}

/\* Move elements from stack1 to stack 2 only if

stack2 is empty \*/

if (q.stack2.isEmpty()) {

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

x = pop(q.stack1);

push(q.stack2, x);

}

}

x = pop(q.stack2);

return x;

}

/\* Driver function to test above functions \*/

public static void main(String args[])

{

/\* Create a queue with items 1 2 3\*/

Queue q = new Queue();

q.stack1 = new Stack<>();

q.stack2 = new Stack<>();

enQueue(q, 3);

enQueue(q, 2);

enQueue(q, 1);

/\* Dequeue items \*/

System.out.print(deQueue(q) + " ");

System.out.print(deQueue(q) + " ");

System.out.println(deQueue(q) + " ");

}

}

**package** linkedlist;

**import** linkedlist.SingularlyLinkedList;

**public** **class** SingularlyLinkedList<E> {

// ---------------- nested Node class ----------------

**private** **static** **class** Node<E> {

**private** E element; // reference to the element stored at this node

**private** Node<E> next; // reference to the subsequent node in the list

**public** Node(E e, Node<E> n) {

element = e;

next = n;

}

**public** E getElement() {

**return** element;

}

**public** Node<E> getNext() {

**return** next;

}

**public** **void** setNext(Node<E> n) {

next = n;

}

} // ----------- end of nested Node class -----------

// instance variables of the SinglyLinkedList

**private** Node<E> head = **null**; // head node of the list (or null if empty)

**private** Node<E> tail = **null**; // last node of the list (or null if empty)

**private** **int** size = 0; // number of nodes in the list

// constructs an initially empty list

**public** SingularlyLinkedList() { }

// access methods

// should return size or number of nodes in the list

// without using 'size' attribute/variable

// hint: use a loop until it reaches null

// Exercise R-3.9

**public** **int** size() {

**int** numberOfNodes = 0;

Node <E> n;

**for** (n = head; n != **null**; n = n.getNext()){

numberOfNodes++;

}

**return** numberOfNodes;

}

// Exercise R-3.12

**public** **void** rotate(){

// codes to rotate list...

// rotate the first element to the back of the list

**if** (tail != **null**) // if empty, do nothing

tail = tail.getNext(); // the old head becomes the new tail

}

**public** **boolean** isEmpty() {

**return** size == 0;

}

**public** E first() {

**if** (isEmpty())

**return** **null**;

**return** head.getElement();

}

**public** E last() {

**if** (isEmpty())

**return** **null**;

**return** tail.getElement();

}

// update methods

**public** **void** addFirst(E e) {

head = **new** Node<>(e, head);

**if** (size == 0)

tail = head;

size++;

}

**public** **void** addLast(E e) {

Node<E> newest = **new** Node<>(e, **null**);

**if** (isEmpty()) // false

head = newest;

**else**

tail.setNext(newest);

tail = newest;

size++;

}

**public** E removeFirst() {

**if** (isEmpty())

**return** **null**;

E answer = head.getElement();

head = head.getNext();

size--;

**if** (size == 0)

tail = **null**;

**return** answer;

}

}

**package** linkedlist;

**import** linkedlist.SingularlyLinkedList;

**public** **class** boxer {

**public** **static** **void** main(String[] args) {

// **TODO** Auto-generated method stub

SingularlyLinkedList<String> boxers= **new** SingularlyLinkedList <String>();

boxers.addFirst("donesia");

boxers.addFirst("balo");

boxers.addFirst("calderon");

boxers.addFirst("jushua");

boxers.removeFirst();

System.***out***.println("size : " + boxers.size());

}

}