## Topics

1. Implement Node Class
2. Generics
3. Implement SinglyLinkedList Class
4. Implement Basic Methods of SinglyLinkedList

* isEmpty()
* size()
* first()
* last()
* addFirst()
* addLast()
* removeFirst()

## Homework

1. develop an implementation of the equals method in the context of the SinglyLinkedList class.

concatenateLists(z, y):

if z is empty:

return y

if yis empty:

return z

current = z.head

while current.next is not null:

current = current.next

current.next = y.head

z' = z

return z'

1. Give an algorithm for finding the second-to-last node in a singly linked list in which the last node is indicated by a null next reference.

public class LinkedList {

private Node head;

public Node findSecondToLastNode() {

if (head == null || head.next == null) {

return null;

}

Node current = head;

Node previous = null;

while (current.next != null) {

pre = current;

current = current.next;

}

return pre;

}

private class Node {

private int data;

private Node next;

public Node(int data) {

this.data = data;

this.next = null;

}

}

}

1. Give an implementation of the size( ) method for the SingularlyLinkedList class, assuming that we did not maintain size as an instance variable.

public class SingularlyLinkedList {

private Node head;

public int size() {

int count = 0;

Node current = head;

while (current != null) {

count++;

current = current.next;

}

return count;

}

private class Node {

private int data;

private Node next;

public Node(int data) {

this.data = data;

this.next = null;

}

}

}

1. Describe an algorithm for concatenating two singly linked lists L and M, into a single list L′ that contains all the nodes of L followed by all the nodes of M.

public class SinglyLinkedList {

private Node head;

public void concatenate(SinglyLinkedList otherList) {

if (head == null) {

head = otherList.head;

} else {

Node current = head;

while (current.next != null) {

current = current.next;

}

current.next = otherList.head;

}

}

private class Node {

private int data;

private Node next;

public Node(int data) {

this.data = data;

this.next = null;

}

}

}

1. Implement a rotate( ) method in the SinglyLinkedList class, which has semantics equal to addLast(removeFirst( )), yet without creating any new node.

public class SinglyLinkedList {

private Node head;

public void rotate() {

if (head == null || head.next == null) {

return;

}

Node previous = null;

Node current = head;

while (current.next != null) {

previous = current;

current = current.next;

}

current.next = head;

head = current;

previous.next = null;

}

private class Node {

private int data;

private Node next;

public Node(int data) {

this.data = data;

this.next = null;

}

}

}

1. Describe in detail an algorithm for reversing a singly linked list L using only a constant amount of additional space.

public class SinglyLinkedList {

private Node head;

public void reverse() {

if (head == null || head.next == null) {

return;

}

Node previous = null;

Node current = head;

while (current != null) {

Node next = current.next;

current.next = previous;

previous = current;

current = next;

}

head = previous;

}

private class Node {

private int data;

private Node next;

public Node(int data) {

this.data = data;

this.next = null;

}

}

}