## Topics

1. Implement Node Class

public class Node<T> {

private T data;

private Node<T> next;

public Node(T data) {

this.data = data;

this.next = null;

}

public T getData() {

return data;

}

public void setData(T data) {

this.data = data;

}

public Node<T> getNext() {

return next;

}

public void setNext(Node<T> next) {

this.next = next;

}

}

1. Generics

public class Box<T> {

private T data;

public void setData(T data) {

this.data = data;

}

public T getData() {

return data;

}

}

1. Implement SinglyLinkedList Class

public class SinglyLinkedList<T> {

private Node<T> head;

private int size;

public SinglyLinkedList() {

head = null;

size = 0;

}

// Other methods will be implemented here (isEmpty, size, first, last, addFirst, addLast, removeFirst)

}

1. Implement Basic Methods of SinglyLinkedList

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

public boolean isEmpty() {

return size == 0;

}

public int size() {

return size;

}

public T first() {

if (isEmpty()) {

return null;

}

return head.getData();

}

public T last() {

if (isEmpty()) {

return null;

}

Node<T> current = head;

while (current.getNext() != null) {

current = current.getNext();

}

return current.getData();

}

public void addFirst(T data) {

Node<T> newNode = new Node<>(data);

newNode.setNext(head);

head = newNode;

size++;

}

public void addLast(T data) {

Node<T> newNode = new Node<>(data);

if (isEmpty()) {

head = newNode;

} else {

Node<T> current = head;

while (current.getNext() != null) {

current = current.getNext();

}

current.setNext(newNode);

}

size++;

}

public T removeFirst() {

if (isEmpty()) {

return null;

}

T removedData = head.getData();

head = head.getNext();

size--;

return removedData;

}

## Homework

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

@Override

public boolean equals(Object obj) {

if (this == obj) {

return true;

}

if (obj == null || getClass() != obj.getClass()) {

return false;

}

SinglyLinkedList<?> otherList = (SinglyLinkedList<?>) obj;

if (size() != otherList.size()) {

return false;

}

Node<T> currentNode = head;

Node<?> otherCurrentNode = otherList.head;

while (currentNode != null) {

if (!currentNode.getData().equals(otherCurrentNode.getData())) {

return false;

}

currentNode = currentNode.getNext();

otherCurrentNode = otherCurrentNode.getNext();

}

return true;

}

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.

function findSecondToLast(head):

current = head

previous = null

while current.getNext() is not null:

previous = current

current = current.getNext()

if previous is null or previous.getPrevious() is null:

return null

else:

return previous

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

public int size() {

int count = 0;

Node<T> current = head;

while (current != null) {

count++;

current = current.getNext();

}

return count;

}

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

public void rotate() {

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

// List is empty or has only one node, no need to rotate

return;

}

Node<T> oldHead = head;

Node<T> current = head;

while (current.getNext() != null) {

current = current.getNext();

}

current.setNext(head.getNext());

head = head.getNext();

current = head;

while (current.getNext() != null) {

current = current.getNext();

}

current.setNext(oldHead);

oldHead.setNext(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.

function concatenateLists(L, M):

if L is empty:

return M

if M is empty:

return L

newHead = L.head

while L.head.next is not null:

L.head = L.head.next

L.head.next = M.head

return newHead

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

function reverseList(L):

current = L.head

previous = null

next = null

while current is not null:

next = current.next

current.next = previous

previous = current

current = next

L.head = previous

return L