**C-DAC Mumbai Date 01/10/2024**

**Subject: Algorithm and Data Structure**

**Assignment 3**

**Solve the assignment with following thing to be added in each question.**

-Program

-Flow chart

-Explanation

-Output

-Time and Space complexity

Submission Date: 3/10/2024

1. **Implement a singly linked list with basic operations: insert, delete, search.**

**public class SinglyLinkedList {**

**private Node head;**

**private static class Node {**

**int data;**

**Node next;**

**Node(int data) {**

**this.data = data;**

**this.next = null;**

**}**

**}**

**public void insert(int data) {**

**Node newNode = new Node(data);**

**if (head == null) {**

**head = newNode;**

**} else {**

**Node current = head;**

**while (current.next != null) {**

**current = current.next;**

**}**

**current.next = newNode;**

**}**

**}**

**public void delete(int data) {**

**if (head == null) return;**

**if (head.data == data) {**

**head = head.next;**

**return;**

**}**

**Node current = head;**

**while (current.next != null) {**

**if (current.next.data == data) {**

**current.next = current.next.next;**

**return;**

**}**

**current = current.next;**

**}**

**}**

**public boolean search(int data) {**

**Node current = head;**

**while (current != null) {**

**if (current.data == data) {**

**return true;**

**}**

**current = current.next;**

**}**

**return false;**

**}**

**public void printList() {**

**Node current = head;**

**while (current != null) {**

**System.out.print(current.data + " ");**

**current = current.next;**

**}**

**System.out.println();**

**}**

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

**SinglyLinkedList list = new SinglyLinkedList();**

**// Test Case 1**

**list.insert(3);**

**list.insert(7);**

**list.insert(5);**

**list.delete(7);**

**System.out.print("List = [");**

**list.printList();**

**System.out.println("Found = " + list.search(5));**

**// Test Case 2**

**list = new SinglyLinkedList();**

**list.insert(9);**

**list.insert(4);**

**list.delete(4);**

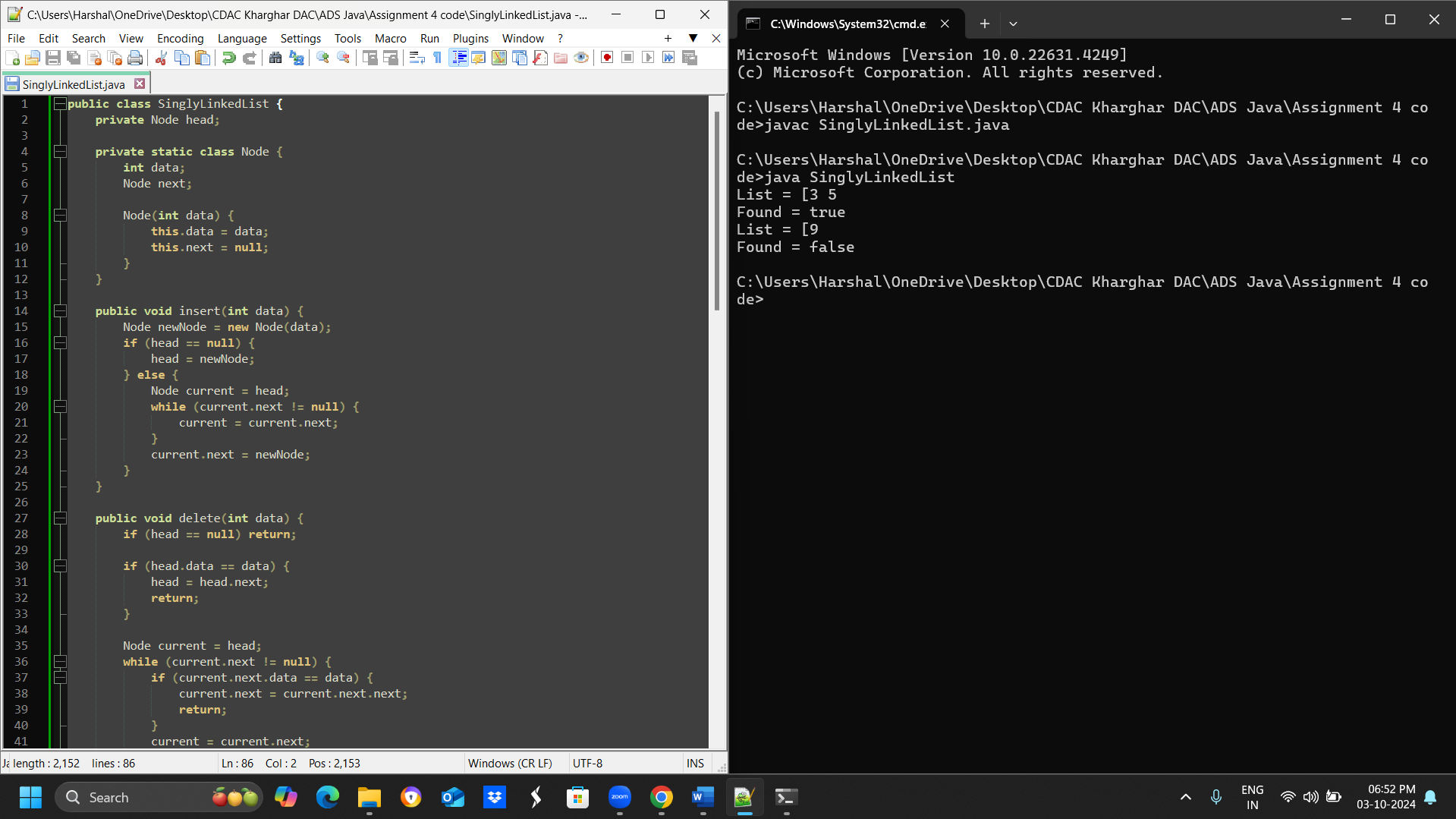
**System.out.print("List = [");**

**list.printList();**

**System.out.println("Found = " + list.search(10));**

**}**

**}**

****

* **Test Case 1**:  
  Input: Insert 3 → Insert 7 → Insert 5 → Delete 7 → Search 5  
  Output: List = [3, 5], Found = True
* **Test Case 2**:  
  Input: Insert 9 → Insert 4 → Delete 4 → Search 10  
  Output: List = [9], Found = False

**2. Reverse a singly linked list.**

* **Test Case 1**:  
  Input: List = [1, 2, 3, 4, 5]  
  Output: List = [5, 4, 3, 2, 1]
* **Test Case 2**:  
  Input: List = [10, 20, 30]  
  Output: List = [30, 20, 10]

1. **Detect a cycle in a linked list.**

**public class LinkedListCycleDetector {**

**public static boolean hasCycle(Node head) {**

**if (head == null) return false;**

**Node slow = head;**

**Node fast = head;**

**while (fast != null && fast.next != null) {**

**slow = slow.next;**

**fast = fast.next.next;**

**if (slow == fast) return true;**

**}**

**return false;**

**}**

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

**// Test Case 1: Cycle Detected**

**Node node1 = new Node(1);**

**Node node2 = new Node(2);**

**Node node3 = new Node(3);**

**Node node4 = new Node(4);**

**Node node5 = new Node(5);**

**node1.next = node2;**

**node2.next = node3;**

**node3.next = node4;**

**node4.next = node5;**

**node5.next = node3; // cycle**

**System.out.println(hasCycle(node1) ? "Cycle Detected" : "No Cycle"); // Output: Cycle Detected**

**// Test Case 2: No Cycle**

**Node node6 = new Node(6);**

**Node node7 = new Node(7);**

**Node node8 = new Node(8);**

**Node node9 = new Node(9);**

**node6.next = node7;**

**node7.next = node8;**

**node8.next = node9;**

**System.out.println(hasCycle(node6) ? "Cycle Detected" : "No Cycle"); // Output: No Cycle**

**}**

**}**

**class Node {**

**int value;**

**Node next;**

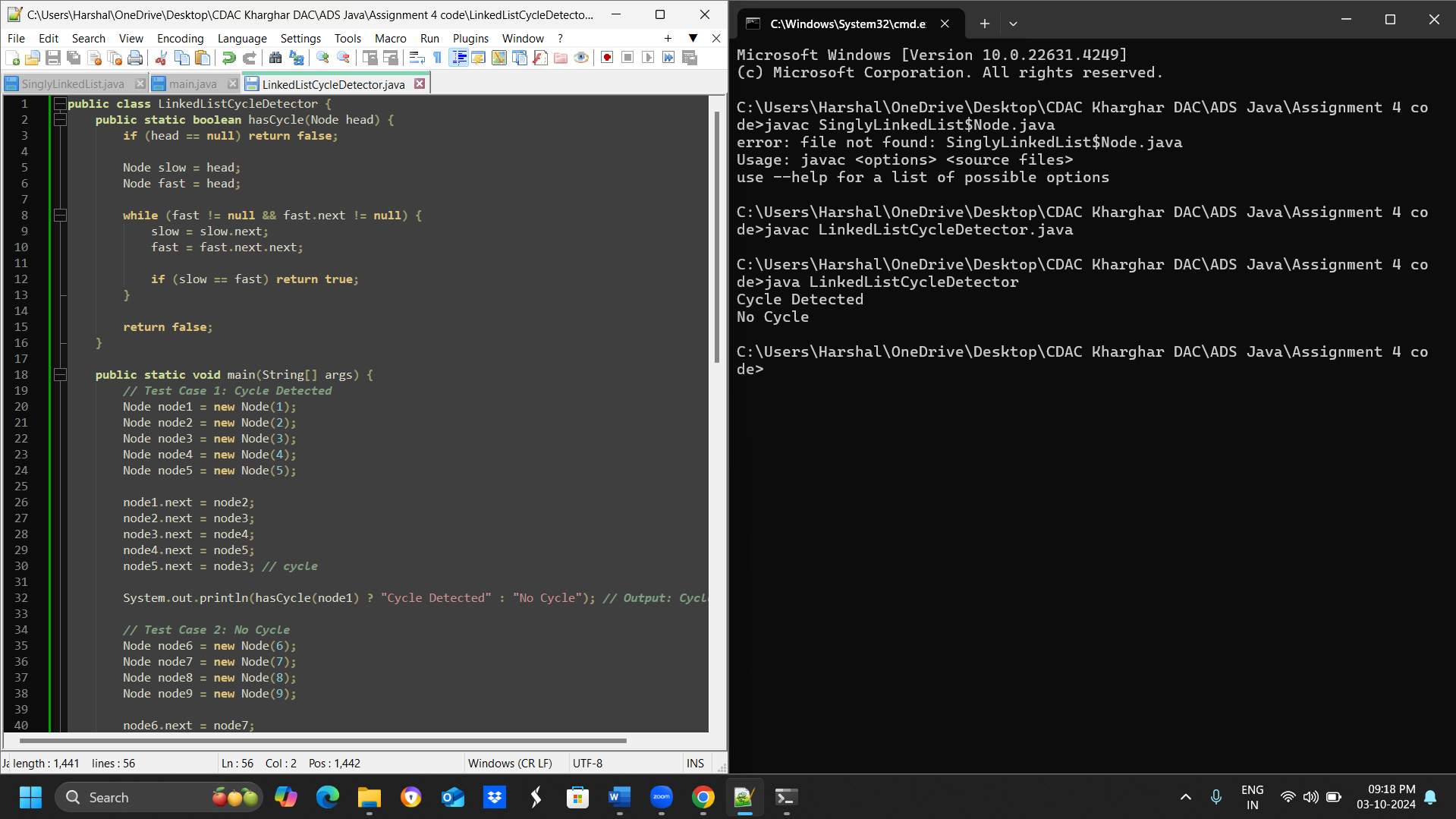
**public Node(int value) {**

**this.value = value;**

**this.next = null;**

**}**

**}**



**Test Case 1**:  
Input: List = [1 → 2 → 3 → 4 → 5 → 3 (cycle)]  
Output: Cycle Detected

* **Test Case 2**:  
  Input: List = [6 → 7 → 8 → 9]  
  Output: No Cycle

**4. Merge two sorted linked lists.**

* **Test Case 1**:  
  Input: List1 = [1, 3, 5], List2 = [2, 4, 6]  
  Output: Merged List = [1, 2, 3, 4, 5, 6]
* **Test Case 2**:  
  Input: List1 = [10, 15, 20], List2 = [12, 18, 25]  
  Output: Merged List = [10, 12, 15, 18, 20, 25]

**5. Find the nth node from the end of a linked list.**

* **Test Case 1**:  
  Input: List = [10, 20, 30, 40, 50], n = 2  
  Output: 40
* **Test Case 2**:  
  Input: List = [5, 15, 25, 35], n = 4  
  Output: 5

**6. Remove duplicates from a sorted linked list.**

* **Test Case 1**:  
  Input: List = [1, 1, 2, 3, 3, 4]  
  Output: List = [1, 2, 3, 4]
* **Test Case 2**:  
  Input: List = [7, 7, 8, 9, 9, 10]  
  Output: List = [7, 8, 9, 10]

**7. Implement a doubly linked list with insert, delete, and traverse operations.**

* **Test Case 1**:  
  Input: Insert 10 → Insert 20 → Insert 30 → Delete 20  
  Output: List = [10, 30]
* **Test Case 2**:  
  Input: Insert 1 → Insert 2 → Insert 3 → Delete 1  
  Output: List = [2, 3]

**8. Reverse a doubly linked list.**

* **Test Case 1**:  
  Input: List = [5, 10, 15, 20]  
  Output: List = [20, 15, 10, 5]
* **Test Case 2**:  
  Input: List = [4, 8, 12]  
  Output: List = [12, 8, 4]

**9. Add two numbers represented by linked lists.**

* **Test Case 1**:  
  Input: List1 = [2 → 4 → 3], List2 = [5 → 6 → 4] (243 + 465)  
  Output: Sum List = [7 → 0 → 8]
* **Test Case 2**:  
  Input: List1 = [9 → 9 → 9], List2 = [1] (999 + 1)  
  Output: Sum List = [0 → 0 → 0 → 1]

**10. Rotate a linked list by k places.**

* **Test Case 1**:  
  Input: List = [10, 20, 30, 40, 50], k = 2  
  Output: List = [30, 40, 50, 10, 20]
* **Test Case 2**:  
  Input: List = [5, 10, 15, 20], k = 3  
  Output: List = [20, 5, 10, 15]

**11. Flatten a multilevel doubly linked list.**

* **Test Case 1**:  
  Input: List = [1 → 2 → 3, 3 → 7 → 8, 8 → 10 → 12]  
  Output: Flattened List = [1 → 2 → 3 → 7 → 8 → 10 → 12]
* **Test Case 2**:  
  Input: List = [1 → 2 → 3, 2 → 5 → 6, 6 → 7 → 9]  
  Output: Flattened List = [1 → 2 → 5 → 6 → 7 → 9 → 3]

**12. Split a circular linked list into two halves.**

* **Test Case 1**:  
  Input: Circular List = [1 → 2 → 3 → 4 → 5 → 6 → (back to 1)]  
  Output: List1 = [1 → 2 → 3], List2 = [4 → 5 → 6]
* **Test Case 2**:  
  Input: Circular List = [10 → 20 → 30 → 40 → (back to 10)]  
  Output: List1 = [10 → 20], List2 = [30 → 40]

**13. Insert a node in a sorted circular linked list.**

**public class CircularLinkedList {**

**Node head;**

**static class Node {**

**int data;**

**Node next;**

**Node(int data) {**

**this.data = data;**

**this.next = null;**

**}**

**}**

**public void insertNode(int newData) {**

**Node newNode = new Node(newData);**

**if (head == null || newData <= head.data) {**

**newNode.next = head;**

**head = newNode;**

**Node temp = head;**

**while (temp.next != head) {**

**temp = temp.next;**

**}**

**temp.next = head;**

**} else {**

**Node current = head;**

**while (current.next != head && current.next.data < newData) {**

**current = current.next;**

**}**

**newNode.next = current.next;**

**current.next = newNode;**

**}**

**}**

**public void printList() {**

**Node temp = head;**

**do {**

**System.out.print(temp.data + " ");**

**temp = temp.next;**

**} while (temp != head);**

**System.out.println();**

**}**

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

**CircularLinkedList list = new CircularLinkedList();**

**// Test Case 1**

**list.head = new Node(10);**

**list.head.next = new Node(20);**

**list.head.next.next = new Node(30);**

**list.head.next.next.next = new Node(40);**

**list.head.next.next.next.next = list.head;**

**System.out.println("Original List: ");**

**list.printList();**

**list.insertNode(25);**

**System.out.println("List after inserting 25: ");**

**list.printList();**

**// Test Case 2**

**list.head = new Node(5);**

**list.head.next = new Node(15);**

**list.head.next.next = new Node(25);**

**list.head.next.next.next = list.head;**

**System.out.println("Original List: ");**

**list.printList();**

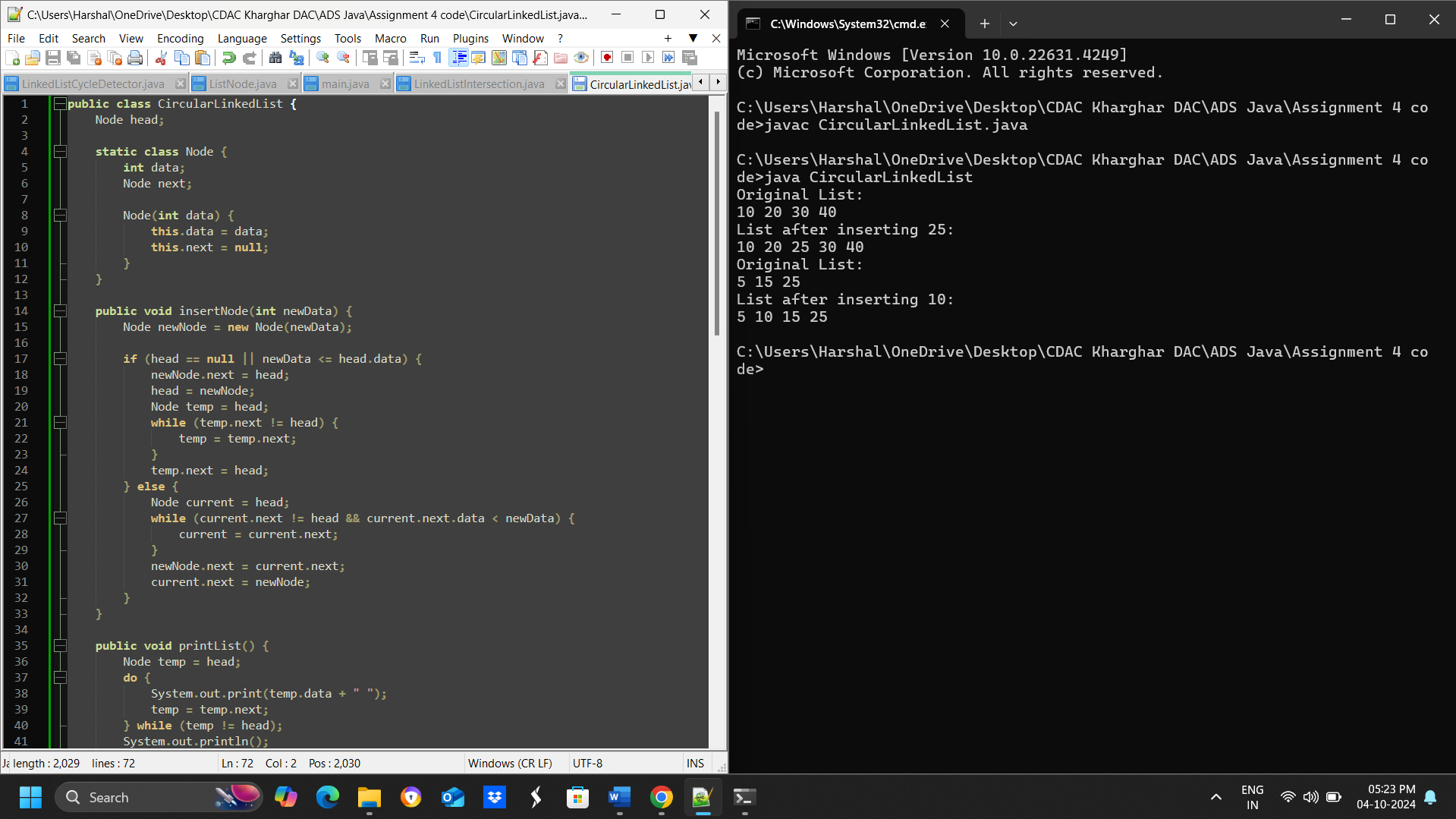
**list.insertNode(10);**

**System.out.println("List after inserting 10: ");**

**list.printList();**

**}**

**}**

****

* **Test Case 1**:  
  Input: Circular List = [10 → 20 → 30 → 40 → (back to 10)], Insert 25  
  Output: Circular List = [10 → 20 → 25 → 30 → 40 → (back to 10)]
* **Test Case 2**:  
  Input: Circular List = [5 → 15 → 25 → (back to 5)], Insert 10  
  Output: Circular List = [5 → 10 → 15 → 25 → (back to 5)]

**14. Check if two linked lists intersect, and find the intersection point if they do.**

**public class LinkedListIntersection {**

**public static Node findIntersection(Node head1, Node head2) {**

**// Calculate the length of both linked lists**

**int len1 = length(head1);**

**int len2 = length(head2);**

**// Move the longer list to make both lists equal in length**

**Node curr1 = head1;**

**Node curr2 = head2;**

**if (len1 > len2) {**

**for (int i = 0; i < len1 - len2; i++) {**

**curr1 = curr1.next;**

**}**

**} else {**

**for (int i = 0; i < len2 - len1; i++) {**

**curr2 = curr2.next;**

**}**

**}**

**// Traverse both lists simultaneously to find the intersection point**

**while (curr1 != null && curr2 != null) {**

**if (curr1 == curr2) {**

**return curr1;**

**}**

**curr1 = curr1.next;**

**curr2 = curr2.next;**

**}**

**// If no intersection is found, return null**

**return null;**

**}**

**public static int length(Node head) {**

**int len = 0;**

**Node curr = head;**

**while (curr != null) {**

**len++;**

**curr = curr.next;**

**}**

**return len;**

**}**

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

**// Test Case 1**

**Node list1 = new Node(1);**

**list1.next = new Node(2);**

**list1.next.next = new Node(3);**

**list1.next.next.next = new Node(4);**

**list1.next.next.next.next = new Node(5);**

**Node list2 = new Node(6);**

**list2.next = new Node(7);**

**list2.next.next = list1.next.next.next; // Intersection point**

**Node intersection = findIntersection(list1, list2);**

**if (intersection != null) {**

**System.out.println("Intersection Point = " + intersection.data);**

**} else {**

**System.out.println("No Intersection");**

**}**

**// Test Case 2**

**Node list3 = new Node(10);**

**list3.next = new Node(20);**

**list3.next.next = new Node(30);**

**list3.next.next.next = new Node(40);**

**Node list4 = new Node(15);**

**list4.next = new Node(25);**

**list4.next.next = new Node(35);**

**intersection = findIntersection(list3, list4);**

**if (intersection != null) {**

**System.out.println("Intersection Point = " + intersection.data);**

**} else {**

**System.out.println("No Intersection");**

**}**

**}**

**}**

**class Node {**

**int data;**

**Node next;**

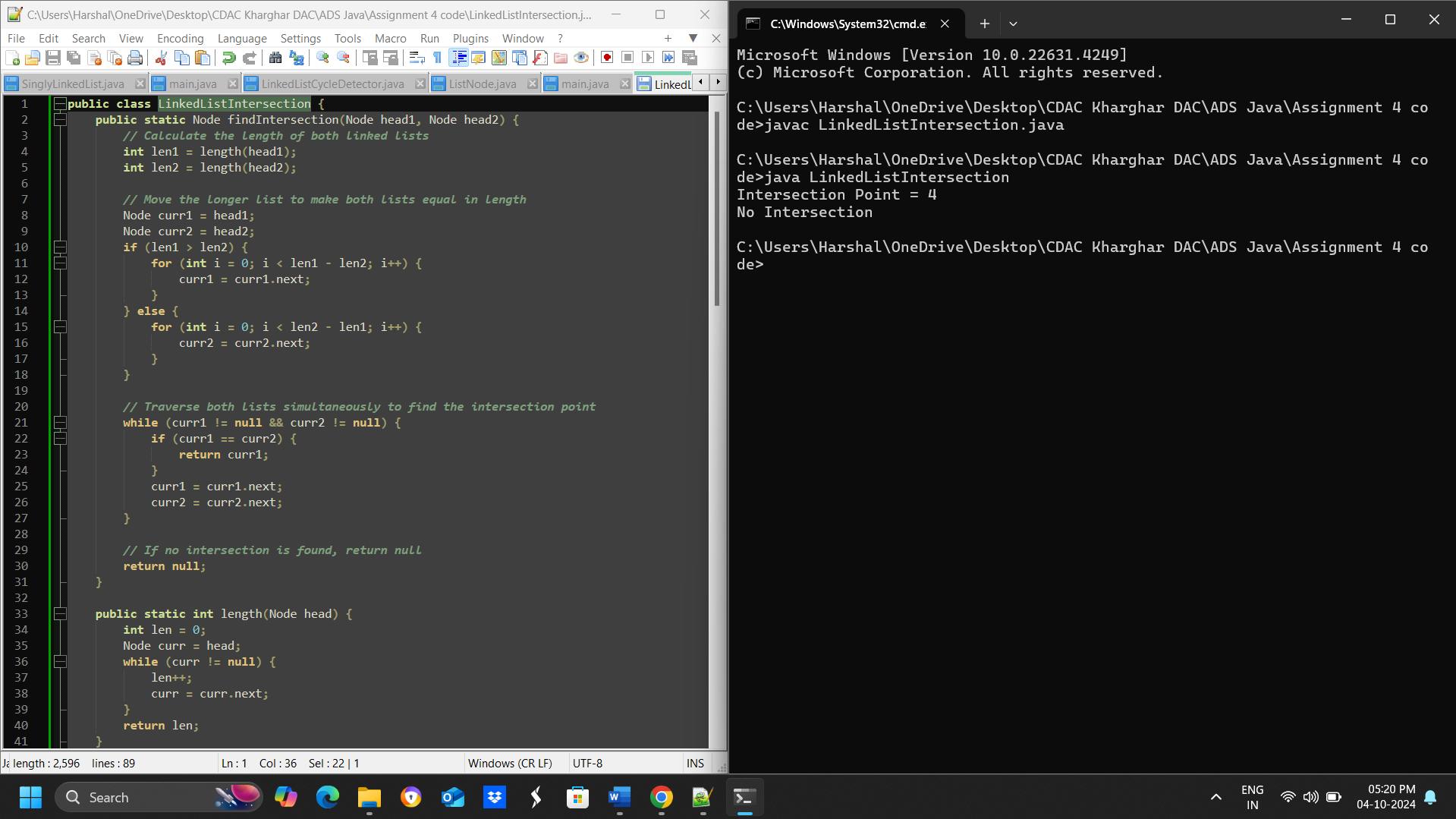
**public Node(int data) {**

**this.data = data;**

**this.next = null;**

**}**

**}**

****

* **Test Case 1**:  
  Input: List1 = [1 → 2 → 3 → 4 → 5], List2 = [6 → 7 → 4 → 5]  
  Output: Intersection Point = 4
* **Test Case 2**:  
  Input: List1 = [10 → 20 → 30 → 40], List2 = [15 → 25 → 35]  
  Output: No Intersection

**15. Find the middle element of a linked list in one pass.**

**// Node class for the linked list**

**class Node {**

**int data;**

**Node next;**

**Node(int data) {**

**this.data = data;**

**this.next = null;**

**}**

**}**

**// Function to find the middle element of the linked list**

**class LinkedList {**

**Node head;**

**// Function to add a new node to the linked list**

**void addNode(int data) {**

**Node newNode = new Node(data);**

**if (head == null) {**

**head = newNode;**

**} else {**

**Node lastNode = head;**

**while (lastNode.next != null) {**

**lastNode = lastNode.next;**

**}**

**lastNode.next = newNode;**

**}**

**}**

**// Function to find the middle element of the linked list**

**int findMiddle() {**

**Node slowPtr = head;**

**Node fastPtr = head;**

**if (head != null) {**

**while (fastPtr != null && fastPtr.next != null) {**

**fastPtr = fastPtr.next.next;**

**slowPtr = slowPtr.next;**

**}**

**}**

**return slowPtr.data;**

**}**

**}**

**public class Main {**

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

**LinkedList list = new LinkedList();**

**list.addNode(1);**

**list.addNode(2);**

**list.addNode(3);**

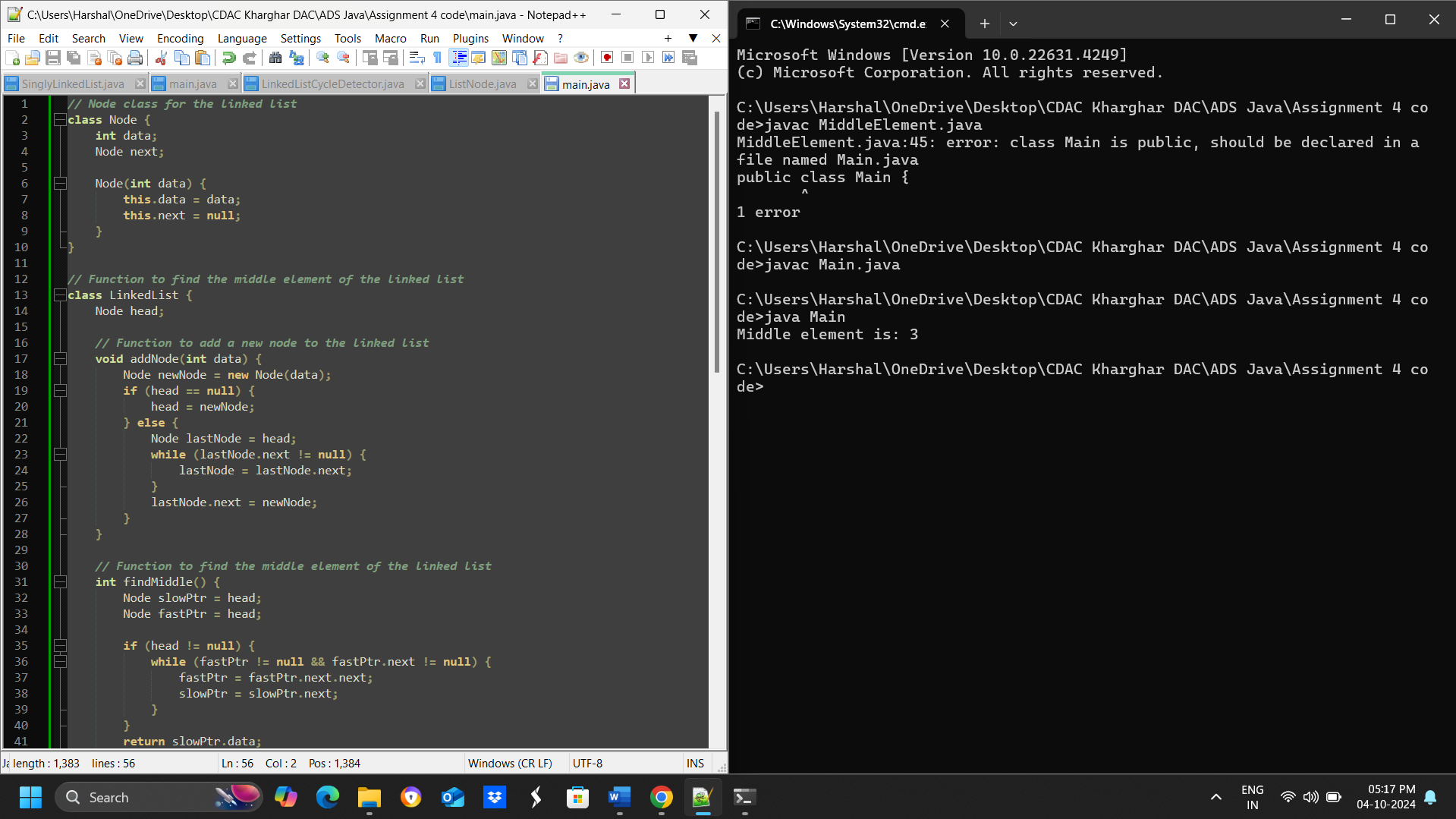
**list.addNode(4);**

**list.addNode(5);**

**System.out.println("Middle element is: " + list.findMiddle());**

**}**

**}**

****

* **Test Case 1**:  
  Input: List = [1, 2, 3, 4, 5]  
  Output: Middle = 3
* **Test Case 2**:  
  Input: List = [11, 22, 33, 44, 55, 66]  
  Output: Middle = 44