Linked List Unit III

Q1 Eliminate duplicates from LL
You have been given a singly linked list of integers where the elements are sorted in ascending order. Write a function that removes the consecutive duplicate values such that the given list only contains unique elements and returns the head to the updated list.
Sample Input :
1 2 3 3 3 3 4 4 4 5 5 7 -1
Sample Output :
1 2 3 4 5 7

Input
1 1 2 3 3 4 5 5 5 6 -1
Output
1 2 3 4 5 6

Input
10 20 20 20 30 40 40 50 -1
Output
10 20 30 40 50

Input
5 5 5 5 5 -1
Output
5

```
import java.util.Scanner;

class Node {
   int data;
   Node next;

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

class LinkedList {
   Node head;

   // Method to insert data into the linked list
   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;
        }
    }

    // Method to remove consecutive duplicates from the linked list
    void removeDuplicates() {
        Node current = head;
        while (current != null && current.next != null) {
            if (current.data == current.next.data) {
                current.next = current.next.next; // Skip the duplicate node
            } else {
                current = current.next; // Move to the next distinct element
            }
        }
    }

    // Method to display the linked list
    void display() {
        Node current = head;
        while (current != null) {
            System.out.print(current.data + " ");
            current = current.next;
        }
        System.out.println();
    }
}

public class Main {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        LinkedList list = new LinkedList();

        // Read input until -1 is encountered
        while (true) {
            int value = scanner.nextInt();
            if (value == -1) {
                break;
            }
            list.insert(value);
        }

        // Remove duplicates
        list.removeDuplicates();
```

```
        // Display the updated list
        list.display();
    }
}
```

Q2 Palindrome LinkedList
You have been given a head to a singly linked list of integers. Write a function check to whether the list given is a 'Palindrome' or not.
Input
9 2 3 3 2 9 -1
Output
True
Input
0 2 3 2 5 -1
Output
False

Input
1 2 3 3 2 1 -1
Output
True

```
import java.util.Scanner;
import java.util.Stack;

class Node {
    int data;
    Node next;

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

class LinkedList {
    Node head;

    // Method to insert data into the linked list
    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;
        }
    }

    // Method to check if the linked list is a palindrome using a stack
    boolean isPalindrome() {
        if (head == null || head.next == null) {
            return true; // An empty list or a single node list is a palindrome
        }

        Stack<Integer> stack = new Stack<>();
        Node current = head;

        // Push all elements onto the stack
        while (current != null) {
            stack.push(current.data);
            current = current.next;
        }

        // Reset current to head for second traversal
        current = head;

        // Compare elements by popping from the stack
        while (current != null) {
            if (current.data != stack.pop()) {
                return false; // Mismatch found
            }
            current = current.next;
        }

        return true; // All elements matched
    }

    // Method to display the linked list
    void display() {
        Node current = head;
        while (current != null) {
            System.out.print(current.data + " ");
            current = current.next;
        }
        System.out.println();
    }
}

public class Main {
```

```
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        LinkedList list = new LinkedList();

        // Read input until -1 is encountered
        while (true) {
            int value = scanner.nextInt();
            if (value == -1) {
                break;
            }
            list.insert(value);
        }

        // Check if the linked list is a palindrome
        boolean result = list.isPalindrome();
        System.out.println(result ? "True" : "False");
    }
}
```

Q3 For a given singly linked list of integers, find and return the node present at the middle of the list.
Note : If the length of the singly linked list is even, then return the first middle node.

Input
1 2 3 4 5 -1
Output
3

Input
1 2 3 4 -1
Output
2

Input
1 2 3 4 5 6 -1
Output
3

```
import java.util.Scanner;

class Node {
    int data;
    Node next;

    Node(int data) {
        this.data = data;
        this.next = null;
```

```
    }
}

class LinkedList {
    Node head;

    // Method to insert data into the linked list
    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;
        }
    }

    // Method to find the middle node of the linked list
    Node findMiddle() {
        if (head == null) {
            return null; // Empty list
        }

        Node slowPointer = head;
        Node fastPointer = head;

        while (fastPointer != null && fastPointer.next != null && fastPointer.next.next != null) {
            slowPointer = slowPointer.next;
            fastPointer = fastPointer.next.next;
        }

        return slowPointer;
    }

    // Method to display the linked list
    void display() {
        Node current = head;
        while (current != null) {
            System.out.print(current.data + " ");
            current = current.next;
        }
        System.out.println();
    }
}
```

```
class Main {
   public static void main(String[] args) {
      Scanner scanner = new Scanner(System.in);
      LinkedList list = new LinkedList();

      // Read input until -1 is encountered
      while (true) {
         int value = scanner.nextInt();
         if (value == -1) {
            break;
         }
         list.insert(value);
      }

      // Find and display the middle node
      Node middleNode = list.findMiddle();
      if (middleNode != null) {
         System.out.println("Middle Node: " + middleNode.data);
      } else {
         System.out.println("The list is empty.");
      }
   }
}
```