Assignment Questions

Q1. Given a linked list and a key ‘X‘ in, the task is to check if X is present in the linked list or not.

Ans :-

```java
import java.util.*;

class Check_Element {
    class Node {
        int data;
        Node next;
        Node(int d) {
        data = d;
        next = null;
        }
    }

    Node head;

    // Inserts a new node at the front of the list
    public void push(int new_data)  {
        Node new_node = new Node(new_data);
        new_node.next = head;
        head = new_node;
    }

    // Checks whether the value x is present in linked list
    public boolean search(Node head, int x)
    {
        Node current = head;
        while (current != null) {
            if (current.data == x)
                return true; // data found
            current = current.next;
        }
        return false; // data not found
    }
    public static void main(String args[]){
        LinkedList llist = new LinkedList();

        llist.push(21);
        llist.push(22);
        llist.push(11);
        llist.push(43);
        llist.push(23);

        if (llist.search(llist.head, 21))
            System.out.println("Yes");
        else
            System.out.println("No");
    }
```

```
}
```

Q2. Insert a node at the given position in a linked list. We are given a pointer to a node, and the new node is inserted after the given node.

Ans : -

```
//node structure

class Node {
    int data;
    Node next;
};

class LinkedList {
  Node head;

  LinkedList(){
    head = null;
  }

  //Add new element at the end of the list
  void push_back(int newElement) {
    Node newNode = new Node();
    newNode.data = newElement;
    newNode.next = null;
    if(head == null) {
      head = newNode;
    } else {
      Node temp = new Node();
      temp = head;
      while(temp.next != null)
        temp = temp.next;
      temp.next = newNode;
    }
  }

  //Inserts a new element at the given position
  void push_at(int newElement, int position) {
    Node newNode = new Node();
    newNode.data = newElement;
    newNode.next = null;

    if(position < 1) {
      System.out.print("\nposition should be >= 1.");
    } else if (position == 1) {
      newNode.next = head;
      head = newNode;
    } else {

      Node temp = new Node();
```

```
      temp = head;
      for(int i = 1; i < position-1; i++) {
        if(temp != null) {
          temp = temp.next;
        }
      }

      if(temp != null) {
        newNode.next = temp.next;
        temp.next = newNode;
      } else {
        System.out.print("\nThe previous node is null.");
      }
    }
  }

  //display the content of the list
  void PrintList() {
    Node temp = new Node();
    temp = this.head;
    if(temp != null) {
      System.out.print("The list contains: ");
      while(temp != null) {
        System.out.print(temp.data + " ");
        temp = temp.next;
      }
      System.out.println();
    } else {
      System.out.println("The list is empty.");
    }
  }

  public static void main(String[] args) {
    LinkedList MyList = new LinkedList();

    //Add three elements at the end of the list.
    MyList.push_back(3);
    MyList.push_back(6);
    MyList.push_back(5);
    MyList.PrintList();

    //Insert an element at position 2
    MyList.push_at(8, 2);
    MyList.PrintList();

    //Insert an element at position 3
    MyList.push_at(9, 3);
    MyList.PrintList();
```

```
  }
}
```

Q3. Given the head of a sorted linked list, delete all duplicates such that each element appears only once. Return the linked list sorted as well.

Ans-
```
//node structure

class Node {
    int data;
    Node next;
};

class LinkedList {
  static Node head;

  LinkedList(){
    head = null;
  }

  //Add new element at the end of the list
  void push_back(int newElement) {
    Node newNode = new Node();
    newNode.data = newElement;
    newNode.next = null;
    if(head == null) {
      head = newNode;
    } else {
      Node temp = new Node();
      temp = head;
      while(temp.next != null)
        temp = temp.next;
      temp.next = newNode;
    }
  }

   static void deleteDuplicates() {
        if(head == null || head.next == null)
            return ;
        Node curr = head;

        while( curr != null && curr.next != null){

            if(curr.data == curr.next.data){
                curr.next = curr.next.next;
            }
            else{
                curr = curr.next;
            }
```

```
        }
        return;
    }
  //display the content of the list
  void PrintList() {
    Node temp = new Node();
    temp = this.head;
    if(temp != null) {
      System.out.print("The list contains: ");
      while(temp != null) {
        System.out.print(temp.data + " ");
        temp = temp.next;
      }
      System.out.println();
    } else {
      System.out.println("The list is empty.");
    }
  }

  public static void main(String[] args) {
    LinkedList MyList = new LinkedList();

    //Add three elements at the end of the list.
    MyList.push_back(3);
    MyList.push_back(3);
    MyList.push_back(5);
    MyList.push_back(6);
    MyList.push_back(6);
    MyList.push_back(7);
    MyList.PrintList();

    deleteDuplicates();

    MyList.PrintList();

  }
}
```

Q4. Given the head of a singly linked list, return true if it is a palindrome or false other>ise.

```
Ans : class LinkedList {

    Node head; // head of list
    Node slow_ptr, fast_ptr, second_half;

    class Node {
        int data;
        Node next;
```

```
        Node(int d)
        {
            data = d;
            next = null;
        }
    }
    boolean isPalindrome(Node head) {
        slow_ptr = head;
        fast_ptr = head;
        Node prev_of_slow_ptr = head;
        Node midnode = null; // To handle odd size list
        boolean res = true; // initialize result

        if (head != null && head.next != null) {

            while (fast_ptr != null
                && fast_ptr.next != null) {
                fast_ptr = fast_ptr.next.next;
                prev_of_slow_ptr = slow_ptr;
                slow_ptr = slow_ptr.next;
            }

            if (fast_ptr != null) {
                midnode = slow_ptr;
                slow_ptr = slow_ptr.next;
            }
            second_half = slow_ptr;
            prev_of_slow_ptr.next = null;
            reverse();
            res = compareLists(head , second_half);
            reverse();

            if (midnode != null) {
                prev_of_slow_ptr.next = midnode;
                midnode.next = second_half;
            }
            else
                prev_of_slow_ptr.next = second_half;
        }
        return res;
    }
    void reverse(){
        Node prev = null;
        Node current = second_half;
        Node next;
        while (current != null) {
            next = current.next;
            current.next = prev;
```

```
            prev = current;
            current = next;
        }
        second_half = prev;
    }

    boolean compareLists(Node head1, Node head2)
    {
        Node temp1 = head1;
        Node temp2 = head2;

        while (temp1 != null && temp2 != null) {
            if (temp1.data == temp2.data) {
                temp1 = temp1.next;
                temp2 = temp2.next;
            }
            else
                return false;
        }

        if (temp1 == null && temp2 == null)
            return true;
        return false;
    }

    public void push(int new_data){
        Node new_node = new Node(new_data);
        new_node.next = head;
        head = new_node;
    }
    public static void main(String[] args){
        LinkedList llist = new LinkedList();

        int arr[] = { 1, 2, 4, 5, 4, 2, 1 };
        for (int i = 0; i < 7; i++) {
            llist.push(arr[i]);
        }
        if (llist.isPalindrome(llist.head) != false) {
            System.out.println("Is Palindrome");
        }
        else {
            System.out.println("Not Palindrome");
        }
    }
}
```

Q5. Given t>o numbers represented by t>o lists, >rite a function that returns the sum list. The sum list is a list representation of the addition of t>o input numbers.

```
class LinkedList {

    static Node head1, head2;

    static class Node {

        int data;
        Node next;
        Node(int d) {
            data = d;
            next = null;
        }
    }

    void addTwoLists(Node first, Node second) {
        Node start1 = new Node(0);
        start1.next = first;
        Node start2 = new Node(0);
        start2.next = second;

        addPrecedingZeros(start1, start2);
        Node result = new Node(0);
        if (sumTwoNodes(start1.next, start2.next, result) == 1) {
            Node node = new Node(1);
            node.next = result.next;
            result.next = node;
        }
        printList(result.next);
    }

    private int sumTwoNodes(Node first, Node second, Node result) {
        if (first == null) {
            return 0;
        }
        int number = first.data + second.data + sumTwoNodes(first.next,
second.next, result);
        Node node = new Node(number % 10);
        node.next = result.next;
        result.next = node;
        return number / 10;
    }
    private void addPrecedingZeros(Node start1, Node start2) {
        Node next1 = start1.next;
        Node next2 = start2.next;
        while (next1 != null && next2 != null) {
```

```
            next1 = next1.next;
            next2 = next2.next;
        }
        if (next1 == null && next2 != null) {
            while (next2 != null) {
                Node node = new Node(0);
                node.next = start1.next;
                start1.next = node;
                next2 = next2.next;
            }
        } else if (next2 == null && next1 != null) {
            while (next1 != null) {
                Node node = new Node(0);
                node.next = start2.next;
                start2.next = node;
                next1 = next1.next;
            }
        }
    }

    void printList(Node head) {
        while (head != null) {
            System.out.print(head.data + " ");
            head = head.next;
        }
        System.out.println("");
    }

    public static void main(String[] args) {
        LinkedList list = new LinkedList();

        // creating first list
        list.head1 = new Node(7);
        list.head1.next = new Node(5);
        list.head1.next.next = new Node(9);
        list.head1.next.next.next = new Node(4);
        list.head1.next.next.next.next = new Node(6);
        System.out.print("First List is ");
        list.printList(head1);

        // creating second list
        list.head2 = new Node(8);
        list.head2.next = new Node(4);
        System.out.print("Second List is ");
        list.printList(head2);

        System.out.print("Resultant List is ");
        // add the two lists and see the result
```

```
        list.addTwoLists(head1, head2);
    }
}
```