Linked List Unit IV

Q Merge Two Sorted LL
You have been given two sorted(in ascending order) singly linked lists of integers.
Write a function to merge them in such a way that the resulting singly linked list is also sorted(in ascending order) and return the new head to the list.

Input
2 5 8 12 -1
3 6 9 -1
Output
2 3 5 6 8 9 12

Input
1 3 5 7 -1
2 4 6 8 -1
Output
1 2 3 4 5 6 7 8

Input
10 20 30 -1
15 25 35 45 -1
Output
10 15 20 25 30 35 45

```
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 merge two sorted linked lists
    static Node mergeSortedLists(Node head1, Node head2) {
        Node dummy = new Node(0); // Dummy node to serve as the start of the merged list
        Node tail = dummy;

        while (head1 != null && head2 != null) {
            if (head1.data <= head2.data) {
                tail.next = head1;
                head1 = head1.next;
            } else {
                tail.next = head2;
                head2 = head2.next;
            }
            tail = tail.next;
        }

        // Append the remaining nodes of either list
        if (head1 != null) {
            tail.next = head1;
        } else {
            tail.next = head2;
        }

        return dummy.next; // The merged list starts from dummy.next
    }

    // 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 list1 = new LinkedList();
        LinkedList list2 = new LinkedList();

        // Read first sorted linked list until -1 is encountered

        while (true) {
```

```
            int value = scanner.nextInt();
            if (value == -1) {
                break;
            }
            list1.insert(value);
        }

        // Read second sorted linked list until -1 is encountered

        while (true) {
            int value = scanner.nextInt();
            if (value == -1) {
                break;
            }
            list2.insert(value);
        }

        // Merge the two sorted linked lists
        Node mergedHead = LinkedList.mergeSortedLists(list1.head, list2.head);

        // Display the merged linked list

        LinkedList mergedList = new LinkedList();
        mergedList.head = mergedHead;
        mergedList.display();
    }
}
```

QEven after Odd LinkedList
For a given singly linked list of integers, arrange the elements such that all the even numbers are placed after the odd numbers. The relative order of the odd and even terms should remain unchanged.
Sample Input
1 4 5 2 -1
Sample Output 1 :
1 5 4 2

Input
1 11 3 6 8 0 9 -1
Output
1 11 3 9 6 8 0

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

```
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 rearrange the linked list so that all odd numbers come before even numbers
    void evenAfterOdd() {
        if (head == null) {
            return;
        }

        Node oddHead = null, oddTail = null;
        Node evenHead = null, evenTail = null;
        Node current = head;

        while (current != null) {
            if (current.data % 2 != 0) { // Odd number
                if (oddHead == null) {
                    oddHead = oddTail = current;
                } else {
                    oddTail.next = current;
                    oddTail = current;
                }
            } else { // Even number
                if (evenHead == null) {
                    evenHead = evenTail = current;
                } else {
                    evenTail.next = current;
                    evenTail = current;
                }
```

```
            }
            current = current.next;
        }

        // If there are no odd numbers, the head should be the start of even list
        if (oddHead == null) {
            head = evenHead;
        } else {
            // Combine odd and even lists
            head = oddHead;
            oddTail.next = evenHead;
        }

        // Ensure the last node points to null
        if (evenTail != null) {
            evenTail.next = null;
        }
    }

    // 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
        //System.out.println("Enter elements of the linked list (end with -1):");
        while (true) {
            int value = scanner.nextInt();
            if (value == -1) {
                break;
            }
            list.insert(value);
        }



        // Rearrange the list so that all odd numbers come before even numbers
        list.evenAfterOdd();

        // Display the modified list
```

```
        list.display();
    }
}
```

Q You have been given a singly linked list of integers along with two integers, 'M,' and 'N.' you need to delete N nodes after every M nodes.

Input
1 2 3 4 5 6 7 8 -1
2 2
Output
1 2 5 6

Input
1 2 3 4 5 6 7 8 -1
2 3
Output
1 2 6 7

Input
10 20 30 40 50 60 70 80 90 -1
2 4
Output
10 20 70 80

```
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 delete N nodes after every M nodes
    void deleteNAfterM(int M, int N) {
        Node current = head;

        while (current != null) {
            // Skip M nodes
            for (int i = 1; i < M && current != null; i++) {
                current = current.next;
            }

            // If we've reached the end of the list, break
            if (current == null) {
                break;
            }

            // Start from the next node and delete N nodes
            Node temp = current.next;
            for (int i = 1; i <= N && temp != null; i++) {
                temp = temp.next;
            }

            // Connect the Mth node to the (M+N+1)th node
            current.next = temp;
            current = temp;
        }
    }

    // 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);
    }

    // Read values for M and N
    int M = scanner.nextInt();
    int N = scanner.nextInt();

    // Delete N nodes after every M nodes
    list.deleteNAfterM(M, N);

    list.display();
  }
}
```