B1) Reverse LinkedList.

```
public class ReverseLinkedList {
    static Node reverse(Node head) {
        Node prev = null;
        Node next = null;
        Node current = head;

        while (current != null) {
            next = current.next;
            current.next = prev;
            prev = current;
            current = next;
        }
        return prev;
    }
}
```

B2) Remove Value

```
public class RemoveValue {
    Node removeValue(Node head, int value) {
        if (head == null) {
            return null;
        }
        while (head != null && head.data.equals(value)) {
            head = head.next;
        }
        Node current = head;
        While (current != null && current.next != null) {
            if (current.next.data.equals(value)) {
                current.next = current.next.next;
            } else {
```

```
            current = current.next;
        }
    }
    return head;
  }
}
```

B3) Multiply Two LinkedList

```
public class MultiplyTwoLinkedList {
    public long multiply(Node headA, Node headB){
        long MOD=1000000007;
        long n1=0;
        long n2=0;
        while (headA != null){
            n1=(n1*10) + headA.data;
            headA= headA.next;
        }
        while (headB != null){
            n2=(n2*10)+ headB.data;
            headB= headB.next;
        }
        return (n1*n2) % MOD;
    }
}
```

B4) Is Palindrome Linked List

```
public class IsPalindromeLinkedList extends ReverseLinkedList{
    static boolean isPalindrome(Node head){
        Node clone = copyNode(head);
        clone = reverse(clone);
        while (clone != null){
```

```
        if (clone.data != head.data ){
            return false;
        }
        clone = clone.next;
        head = head.next;
    }
    return true;
}

static Node copyNode(Node head) {
    if(head == null){
        return null;
    }
    Node copyHead = new Node(head.data);
    Node current = head.next.
    Node copyCurrent = copyHead;

    while(current != null ) {
        copyCurrent.next = new Node(current.data);
        copyCurrent. = copyCurrent.next;
        current = current.next;
    }
    return copyHead;
}
}
```

B5) Merge Two Sorted Linked List.

```
public class MergeTwoSortedLinkedList {
    static Node merge(Node head1, Node head2) {
        if(head1 == null){ return head2;}
        else if (head2 == null) { return head1;}
```

```
Node dummy = new Node(0);
Node current = dummy;

while(head1 != null && head2 != null){
    if(head1.data < head2.data){
        current.next = head1;
        head1 = head1.next;
    }else{
        current.next = head2;
        head2 = head2.next;
    }
    current = current.next;
}
if(head1 != null){
    current.next = head1;
}
if(head2 != null){
    current.next = head2;
}
return dummy.next;
}
```

B6) Remove Nth Node From End

```
public class RemoveNthNodeFromEnd {
    static Node remove(Node head, int n){
        Node dummy = new Node(0);
        dummy.next = head;
        Node iterator1 = dummy;
        Node iterator2 = dummy;

        for(int i=1; i<=n; i++){
```

```
        iterator1 = iterator1.next;
    }
    while (iterator1 != null && iterator1.next != null) {
        iterator1 = iterator1.next;
        iterator2 = iterator2.next;
    }
    iterator2.next = iterator2.next.next;

    return dummy.next;
  }
}
```

B7) Delete Middle of LinkedList

```
public class DeleteMiddleOfLinkedList {
    static Node delete(Node head) {
        if (head == null || head.next == null) {
            return head;
        }
        Node temp1 = head;
        Node temp2 = head;
        int length = 0;
        while (temp1 != null) {
            length++;
            temp1 = temp1.next;
        }
        int mid = length / 2;
        for (int i = 1; i < mid; i++) {
            temp2 = temp2.next;
        }
        temp2.next = temp2.next.next;
        return head;
    }
}
```