1290. Convert Binary Number in a Linked List to Integer

```
class Solution {
    public int getDecimalValue(ListNode head) {
        String s = "";
        while(head!=null){
            s= s+Integer.toString(head.val);
            head = head.next;

        }
       return Integer.parseInt(s,2);
    }
}
```

876. Middle of the Linked List

```
class Solution {
    public ListNode middleNode(ListNode head) {
        ListNode fast = head, slow = head;
        while(fast !=null && fast.next !=null){
            slow = slow.next;
            fast = fast.next.next;
        }
        return slow;
    }
}
```

237. Delete Node in a Linked List

```
class Solution {
    public void deleteNode(ListNode node) {
        node.val = node.next.val;
        node.next = node.next.next;

    }
}
```

21. Merge Two Sorted Lists

```
class Solution {
    public ListNode mergeTwoLists(ListNode list1, ListNode list2) {
        ListNode tempnode = new ListNode(0);
        ListNode current_node = tempnode;
        while(list1 !=null && list2 != null){
            if(list1.val < list2.val){
                current_node.next = list1;
                list1 = list1.next;
            }else{
                current_node.next = list2;
                list2 = list2.next;
            }
            current_node = current_node.next;
        }
        if(list1 != null){
            current_node.next = list1;
            list1 = list1.next;
        }
        if(list2 != null){
            current_node.next = list2;
            list2 = list2.next;
        }
        return tempnode.next;
    }
}
```

83. Remove Duplicates from Sorted List

```
class Solution {
    public ListNode deleteDuplicates(ListNode head) {
        ListNode temp = head;
        while(temp != null){
            if(temp.next == null){
                break;
            }
            if(temp.val == temp.next.val){
                temp.next = temp.next.next;
            }
            else{
                temp = temp.next;
            }
        }
        return head;
    }
}
```

234. Palindrome Linked List

```
class Solution
{
    public boolean isPalindrome(ListNode head)
    {

        ListNode cur = head;
        ListNode middleNode = getMiddle(head);
        ListNode anotherHalf = reverseLinkedList(middleNode);

        while(anotherHalf != null)
        {
            if(cur.val == anotherHalf.val)
            {
                cur = cur.next;
                anotherHalf = anotherHalf.next;
            }
            else
                return false;
        }
        return true;
    }

    public ListNode getMiddle(ListNode head)
    {
        ListNode slow = head;
        ListNode fast = head;

        while(fast != null && fast.next != null)
        {
            slow = slow.next;
            fast = fast.next.next;
        }
        return slow;
    }

    public ListNode reverseLinkedList(ListNode head)
    {
        ListNode prev = null;
        ListNode cur = head;
        ListNode next_node = cur.next;

        while(cur != null)
        {
            next_node = cur.next;
            cur.next = prev;
            prev = cur;
            cur = next_node;
        }
        return prev;
    }
}


public class Solution {
    public boolean hasCycle(ListNode head) {
        if(head == null || head.next == null)
            return false;
        ListNode slow = head;
        ListNode fast = head;

        while(fast.next != null && fast.next.next != null){
            slow = slow.next;
            fast = fast.next.next;
            if(fast == slow)
                return true;
        }
        return false;

    }
}
```

203. Remove Linked List Elements

```
class Solution {
    public ListNode removeElements(ListNode head, int val) {
        while(head !=null && head.val == val){
            head = head.next;
        }
        ListNode CurrentNode = head;
        while(CurrentNode != null && CurrentNode.next!=null){
            if(CurrentNode.next.val == val){
                CurrentNode.next = CurrentNode.next.next;
            }
            else{
                CurrentNode = CurrentNode.next;
            }
        }
        return head;
    }
}
```

2181. Merge Nodes in Between Zeros

```
class Solution {
    public ListNode mergeNodes(ListNode head) {
                    ListNode dummyHead = new ListNode(-1);
                    ListNode newIt = dummyHead;
                    ListNode it = head;
                    while(it!=null){
                            it = it.next;
                            int sumNodes = 0;
                            while(it!=null && it.val!=0){
                                    sumNodes +=  it.val;
                                    it = it.next;
                            }

                            if(sumNodes!=0) {
                                    newIt.next = new ListNode(sumNodes);
                                    newIt = newIt.next;
                            }
                    }
                    return dummyHead.next;
            }
}
```

```
class Solution {
    public ListNode oddEvenList(ListNode head) {
        if(head == null)
            return head;
        ListNode oddHead = head;
        ListNode evenHead = head.next;
        ListNode even = evenHead;


        while(evenHead != null && evenHead.next != null){
            oddHead.next = evenHead.next;
            oddHead = oddHead.next;
            evenHead.next = oddHead.next;
            evenHead = evenHead.next;
        }
        oddHead.next = even;

        return head;

    }
}
```

Merge In Between Linked Lists

```
class Solution {
    public ListNode mergeInBetween(ListNode list1, int a, int b, ListNode list2) {
        int count = 0;
        ListNode head = list1;
        while(list1.next!=null && count<a-1){
            list1 = list1.next;
            count++;
        }
        ListNode temp = list1;
        while(temp!=null && count<b){
            temp = temp.next;
            count++;
        }

        list1.next = list2;
        while(list1.next!=null)
            list1 = list1.next;

        list1.next = temp.next;

        return head;
    }

}
```

2130. Maximum Twin Sum of a Linked List

```
class Solution {
    public int pairSum(ListNode head) {
        Stack<ListNode> st=new Stack();
        ListNode temp = head;
        while(temp != null){
            st.push(temp);
            temp = temp.next;
        }
        int max=Integer.MIN_VALUE;
        temp = head;
        int size=st.size();
        while(st.size()>size/2){
            int first = temp.val;
            int last = st.pop().val;
            max=Math.max(max,(first+last));
            temp = temp.next;
        }
        return max;

    }
}
```