Linked List: Leetcode

(Part 1)

Q206. Reverse Linked List

```
class Solution {
public:
    ListNode* reverseList(ListNode* head) {
        ListNode* prev = nullptr;
        while (head) {
            ListNode* nextNode = head->next;
            head->next = prev;
            prev = head;
            head = nextNode;
        }
        return prev;
    }
};
```

Q21. Merge Two Sorted Lists

```
class Solution {
public:
    ListNode* mergeTwoLists(ListNode* l1, ListNode* l2) {
        ListNode dummy, *tail = &dummy;
        while (l1 && l2) {
            if (l1->val < l2->val) {
                tail->next = l1;
                l1 = l1->next;
            } else {
                tail->next = l2;
                l2 = l2->next;
            }
            tail = tail->next;
        }
        tail->next = l1 ? l1 : l2;
        return dummy.next;
    }
};
```

Q19. Remove N-th Node From End of List

```
class Solution {
public:
    ListNode* removeNthFromEnd(ListNode* head, int n) {
        ListNode dummy(0, head);
        ListNode* fast = &dummy, *slow = &dummy;
        for (int i = 0; i < n; ++i) fast = fast->next;
        while (fast->next) {
            fast = fast->next;
            slow = slow->next;
        }
        slow->next = slow->next->next;
        return dummy.next;
    }
};
```

Q876. Middle of the Linked List

```
class Solution {
public:
    ListNode* middleNode(ListNode* head) {
        ListNode* slow = head, *fast = head;
        while (fast && fast->next) {
            slow = slow->next;
            fast = fast->next->next;
        }
        return slow;
    }
};
```

Q237. Delete Node in a Linked List

```
class Solution {
public:
    void deleteNode(ListNode* node) {
        node->val = node->next->val;
        node->next = node->next->next;
    }
};
```

Q203. Remove Linked List Elements

```
class Solution {
public:
    ListNode* removeElements(ListNode* head, int val) {
        ListNode dummy(0, head), *cur = &dummy;
        while (cur->next) {
            if (cur->next->val == val)
                cur->next = cur->next->next;
            else
                cur = cur->next;
        }
        return dummy.next;
    }
};
```

Q83. Remove Duplicates from Sorted List

```
class Solution {
public:
    ListNode* deleteDuplicates(ListNode* head) {
        ListNode* curr = head;
        while (curr && curr->next) {
            if (curr->val == curr->next->val)
                curr->next = curr->next->next;
            else
                curr = curr->next;
        }
        return head;
    }
};
```

Q82. Remove Duplicates from Sorted List II

```
class Solution {
public:
    ListNode* deleteDuplicates(ListNode* head) {
        ListNode dummy(0, head), *prev = &dummy;
        while (head) {
            if (head->next && head->val == head->next->val)
{
                while (head->next && head->val ==
head->next->val)
                    head = head->next;
                prev->next = head->next;
            } else {
                prev = prev->next;
            }
            head = head->next;
        }
        return dummy.next;
    }
};
```

Q24. Swap Nodes in Pairs

```
class Solution {
public:
    ListNode* swapPairs(ListNode* head) {
        ListNode dummy(0, head), *prev = &dummy;
        while (head && head->next) {
            ListNode* a = head;
            ListNode* b = head->next;
            prev->next = b;
            a->next = b->next;
            b->next = a;
            prev = a;
            head = a->next;
        }
        return dummy.next;
    }
};
```

Q61. Rotate List

```
class Solution {
public:
    ListNode* rotateRight(ListNode* head, int k) {
        if (!head || !head->next || k == 0) return head;
        ListNode* curr = head;
        int len = 1;
        while (curr->next) {
            curr = curr->next;
            len++;
        }
        curr->next = head;
        k %= len;
        for (int i = 0; i < len - k; ++i) curr = curr->next;
        head = curr->next;
        curr->next = nullptr;
        return head;
    }
};
```