Linked List: Leetcode

(Part 2)

Q328. Odd Even Linked List

```
class Solution {
public:
    ListNode* oddEvenList(ListNode* head) {
        if (!head) return nullptr;
        ListNode* odd = head, *even = head->next, *evenHead
= even;
        while (even && even->next) {
            odd->next = even->next;
            odd = odd->next;
            even->next = odd->next;
            even = even->next;
        }
        odd->next = evenHead;
        return head;
    }
};
```

Q86. Partition List

```
class Solution {
public:
    ListNode* partition(ListNode* head, int x) {
        ListNode beforeHead(0), afterHead(0);
        ListNode* before = &beforeHead, *after = &afterHead;
        while (head) {
            if (head->val < x) {
                before->next = head;
                before = before->next;
            } else {
                after->next = head;
                after = after->next;
            }
            head = head->next;
        }
        after->next = nullptr;
        before->next = afterHead.next;
        return beforeHead.next;
    }
};
```

Q143. Reorder List

```
class Solution {
public:
    void reorderList(ListNode* head) {
        if (!head || !head->next) return;
        ListNode* slow = head, *fast = head;
        while (fast && fast->next) {
            slow = slow->next;
            fast = fast->next->next;
        }
        ListNode* prev = nullptr, *curr = slow->next;
        slow->next = nullptr;
        while (curr) {
            ListNode* nextNode = curr->next;
            curr->next = prev;
            prev = curr;
            curr = nextNode;
        }
        ListNode* first = head, *second = prev;
        while (second) {
            ListNode* tmp1 = first->next, *tmp2 =
second->next;
            first->next = second;
            second->next = tmp1;
            first = tmp1;
            second = tmp2;
        }
    }
};
```

Q92. Reverse Linked List II

```
class Solution {
public:
    ListNode* reverseBetween(ListNode* head, int left, int
right) {
        ListNode dummy(0, head), *prev = &dummy;
        for (int i = 1; i < left; ++i) prev = prev->next;
        ListNode* curr = prev->next;
        for (int i = 0; i < right - left; ++i) {
            ListNode* temp = curr->next;
            curr->next = temp->next;
            temp->next = prev->next;
            prev->next = temp;
        }
        return dummy.next;
    }
};
```

Q234. Palindrome Linked List

```
class Solution {
public:
    bool isPalindrome(ListNode* head) {
        ListNode* slow = head, *fast = head;
        while (fast && fast->next) {
            slow = slow->next;
            fast = fast->next->next;
        }
        ListNode* prev = nullptr;
        while (slow) {
            ListNode* nextNode = slow->next;
            slow->next = prev;
            prev = slow;
            slow = nextNode;
        }
        while (prev) {
            if (head->val != prev->val) return false;
            head = head->next;
            prev = prev->next;
        }
        return true;
    }
};
```

Q160. Intersection of Two Linked Lists

```
class Solution {
public:
    ListNode* getIntersectionNode(ListNode* a, ListNode* b)
{
        ListNode* p1 = a, *p2 = b;
        while (p1 != p2) {
            p1 = p1 ? p1->next : b;
            p2 = p2 ? p2->next : a;
        }
        return p1;
    }
};
```

Q141. Linked List Cycle

```cpp
class Solution {
public:
    bool hasCycle(ListNode* head) {
        ListNode* slow = head, *fast = head;
        while (fast && fast->next) {
            slow = slow->next;
            fast = fast->next->next;
            if (slow == fast) return true;
        }
        return false;
    }
};
```

Q142. Linked List Cycle II

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

Q25. Reverse Nodes in k-Group

```
class Solution {
public:
    ListNode* reverseKGroup(ListNode* head, int k) {
        ListNode* ptr = head;
        for (int i = 0; i < k; ++i) {
            if (!ptr) return head;
            ptr = ptr->next;
        }
        ListNode* prev = reverseKGroup(ptr, k);
        while (k--) {
            ListNode* nextNode = head->next;
            head->next = prev;
            prev = head;
            head = nextNode;
        }
        return prev;
    }
};
```

Q725. Split Linked List in Parts

```
class Solution {
public:
    vector<ListNode*> splitListToParts(ListNode* head, int
k) {
        int n = 0;
        for (ListNode* curr = head; curr; curr = curr->next)
++n;
        int len = n / k, rem = n % k;
        vector<ListNode*> res;
        for (int i = 0; i < k; ++i) {
            ListNode* partHead = head, *prev = nullptr;
            for (int j = 0; j < len + (i < rem); ++j) {
                prev = head;
                head = head->next;
            }
            if (prev) prev->next = nullptr;
            res.push_back(partHead);
        }
        return res;
    }
};
```