Leetcode 141 - Detect Loop In Linked List

```
class Solution {
public:
    bool hasCycle(ListNode *head) {
        ListNode slow=head;
        ListNode fast=head;

        while(fast && fast->next){
            slow=slow.next;
            fast=fast.next.next;

            if(slow==fast){
                return true;
            }
        }
        return false;

    }
};
```

Leetcode 206 - Reverse Linked List

```
class Solution {
public:
    ListNode* reverseList(ListNode* head) {
        ListNode* prev = nullptr;
        ListNode* curr = head;

        while (curr != nullptr) {
            ListNode* nextNode = curr->next; // store next node
            curr->next = prev;               // reverse the link
            prev = curr;                     // move prev forward
            curr = nextNode;                 // move curr forward
```

```
        }

        return prev; // new head
    }
};
```

Leetcode 876 - Middle Of The Linked List

```
class Solution {
public:
    ListNode* middleNode(ListNode* head) {

        ListNode* slow=head;
        ListNode* fast=head;

        while(fast&& fast->next){
            slow=slow->next;
            fast=fast->next->next;
        }
        return slow;
    }
};
```

Leetcode 83 - Remove Duplocates

```
class Solution {
public:
    ListNode* deleteDuplicates(ListNode* head) {
        ListNode* current = head;

        while (current != nullptr && current->next != nullptr) {
            if (current->val == current->next->val) {
                current->next = current->next->next;
            } else {
```

```
                current = current->next;
            }
        }

        return head;
    }
};
```

Leetcode 237 - Delete Node

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

Leetcode 234 - Palindrom Linked List

```
class Solution {
public:
    bool isPalindrome(ListNode* head) {

        // step1 Find middle element
        ListNode* slow = head;
        ListNode* fast = head;

        while (fast && fast->next) {
            slow = slow->next;
            fast = fast->next->next;
```

```
        }

        //step2 reverse a linked list

        ListNode* prev=nullptr;
        ListNode* curr=slow;

        while(curr){
            ListNode* nextNode=curr->next;
            curr->next=prev;
            prev=curr;
            curr=nextNode;
        }


        //compare both
        ListNode* first=head;
        ListNode* second=prev;

        while(second){
            if(first->val!=second->val)
            return false;
            first=first->next;
            second=second->next;
        }
        return true;
    }
};
```

Leetcode 2130 - Maximum Twin Sum

```
class Solution {
public:
    int pairSum(ListNode* head) {
        // Step 1: Find middle of the list
        ListNode* slow = head;
```

```
        ListNode* fast = head;

        while (fast && fast->next) {
            slow = slow->next;
            fast = fast->next->next;
        }

        // Step 2: Reverse the second half
        ListNode* prev = nullptr;
        ListNode* curr = slow;
        while (curr) {
            ListNode* nextNode = curr->next;
            curr->next = prev;
            prev = curr;
            curr = nextNode;
        }

        // Step 3: Calculate twin sums
        int maxSum = 0;
        ListNode* first = head;
        ListNode* second = prev;

        while (second) {
            int sum = first->val + second->val;
            maxSum = max(maxSum, sum);
            first = first->next;
            second = second->next;
        }

        return maxSum;
    }
};
```

Leetcode 86 - Partition List

```
class Solution {
```

```
public:
    ListNode* partition(ListNode* head, int x) {
        // Dummy nodes to create two separate lists
        ListNode* smallerDummy = new ListNode(0);
        ListNode* greaterDummy = new ListNode(0);

        // Pointers to build the two lists
        ListNode* smaller = smallerDummy;
        ListNode* greater = greaterDummy;

        // Traverse the original list
        while (head != NULL) {
            if (head->val < x) {
                // Add to 'smaller' list
                smaller->next = head;
                smaller = smaller->next;
            } else {
                // Add to 'greater or equal' list
                greater->next = head;
                greater = greater->next;
            }
            head = head->next;
        }

        // End the greater list
        greater->next = NULL;

        // Connect the two lists
        smaller->next = greaterDummy->next;

        // Return the head of the new list (skipping dummy)
        return smallerDummy->next;
    }
};
```

Leetcode 328 - Odd Even Linked List

```
class Solution {
public:
    ListNode* oddEvenList(ListNode* head) {
        if (head == NULL || head->next == NULL) {
            return head;
        }

        ListNode* odd = head;
        ListNode* even = head->next;
        ListNode* evenHead = even;

        while (even != NULL && even->next != NULL) {
            odd->next = even->next;
            odd = odd->next;

            even->next = odd->next;
            even = even->next;
        }

        odd->next = evenHead;

        return head;
    }
};
```