Leetcode 876

```
struct ListNode * middleNode(struct ListNode * head){
    struct ListNode *slow = head;
    struct ListNode * fast = head;
    while(fast != NULL && fast->next != NULL){
        slow = slow->next;
        fast = fast->next->next;
    }
    return slow;
}
```

O/P Input

head = [1,2,3,4,5]

O/P [3,4,5]

Expected [3,4,5]

Leetcode 203

```
struct ListNode* removeElements(struct ListNode* head, int val){
    struct ListNode dummy;
    dummy.next = head;
    struct ListNode * current = &dummy;
    while(current->next != NULL){
        if(current->next->val == val){
            current->next = current->next->next;
        } else {
            current = current->next;
        }
    }
    return dummy.next;
}
```

o/p Input =

head = [1, 2, 6, 3, 4, 5, 6]

val = 6

o/p [1, 2, 3, 4, 5]

Expected [1, 2, 3, 4, 5]

Leetcode 141

```
bool hasCycle(struct ListNode *head) {
    struct ListNode *slow = head;
    struct ListNode *fast = head;
    while (fast != NULL && fast->next != NULL) {
        slow = slow->next;
        fast = fast->next->next;
        if (slow == fast) {
            return true;
        }
    }
    return false;
}
```

o/p Input

head = [3, 2, 0, -4]

pos = 1

o/p true

exp: true

Leetcode 617

```
struct TreeNode * mergeTrees (struct TreeNode * root1, struct
                              TreeNode * root2){

    if (root1 == NULL)
        return root2;
    if (root2 == NULL)
        return root1;

    struct TreeNode * merged = malloc (sizeof (struct TreeNode));
    merged->val = root1->val + root2->val;
    merged->left = mergeTrees (root1->left, root2->left);
    merged->right = mergeTrees (root1->right, root2->right);
    return merged;
}
```

Inp Input

root1 = [1, 3, 2, 5]

root2 = [2, 1, 3, null, 4, null, 7]

Op e [3, 4, 5, 5, 4, null, 7]

Exp [3, 4, 5, 5, 4, null, 7]