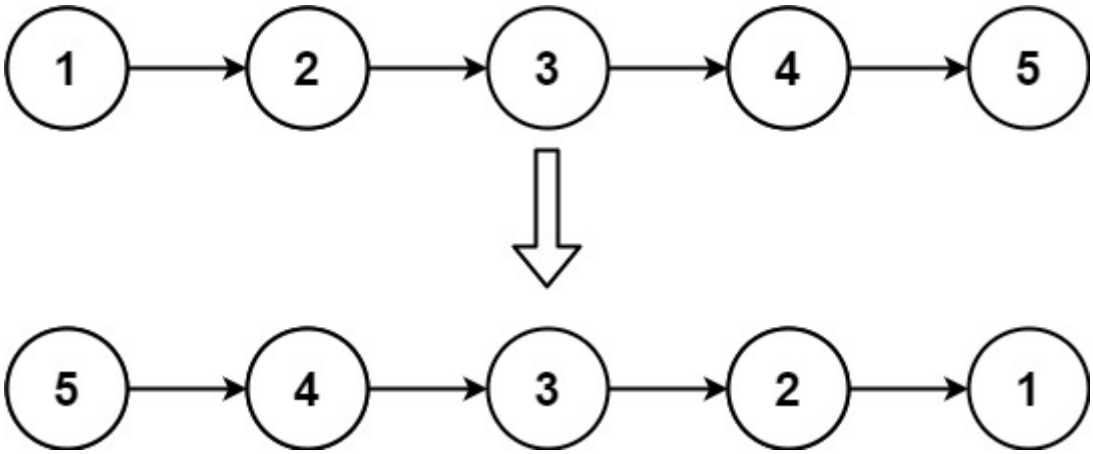
Striver's SDE Sheet

Day 5: Linked List

Reverse Linked List

Given the head of a singly linked list, reverse the list, and return *the reversed list*.



Ans:

Observation and approach : process based question

In first go brute force came in my mind is to make new list insert the list reverse manner like first make a node with same value as first then make a another node and make its head an link the old one and so on

Code: //same thing cam be done with help of recursion

```
ListNode* reverseList(ListNode* head) {
        ListNode* prev = NULL;
        ListNode* curr = head;
        while(curr != NULL){
            ListNode* forward = curr->next;
            curr->next = prev;
            prev = curr;
            curr = forward;
        }
        return prev; }
```

a simple variation that can we solved easily by doing process

Middle of the Linked List

Given the head of a singly linked list, return *the middle node of the linked list*.

If there are two middle nodes, return the second middle node.

Ans:

Observation and approach : process based + some tips & tricks

In first go brute force will be find length by traveling whole linked list then again travel to n/2 length

Code:

```
int length(ListNode* head){
        int len = 0;
        while(head != NULL)
        { len++;
          head = head->next;
        }
        return len;
    }
    ListNode* middleNode(ListNode* head) {
        int len = length(head);
        int ans = len/2;
        ListNode* temp = head;
        int cnt = 0;
        while(cnt < ans)
        {
            temp = temp->next;
            cnt++;
        }
        return temp;
    }
```

Second approach will be of fast and slow pointer method which is an very important method in which we used concept of speed and distance , our slow pointer travel with speed 1 and fast pointer move with speed 2 when they meet at same point then its always a middle point

Code:

```
  ListNode* slow = head;
        ListNode* fast = head;
         while (fast)
         {
             fast = fast->next;
             if (fast)
             {
                 fast=fast->next;
                 slow = slow->next;
             }
         }
         return slow;
```

no big variation possible

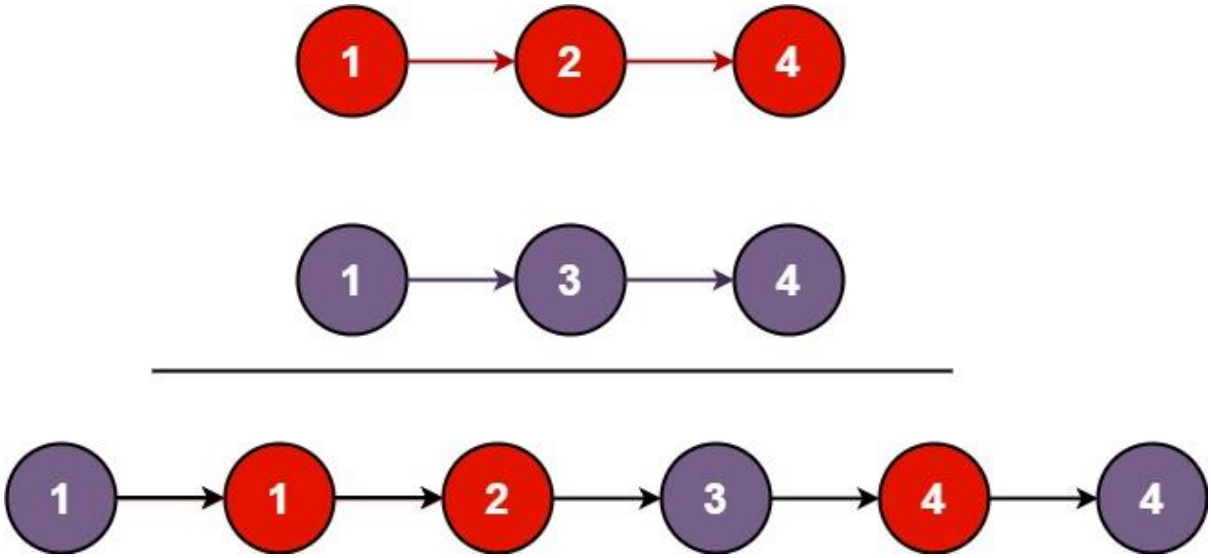
Merge Two Sorted Lists

You are given the heads of two sorted linked lists list1 and list2.

Merge the two lists in a one sorted list. The list should be made by splicing together the nodes of the first two lists.

Return *the head of the merged linked list*.

Ans:



Observation and approach: process based question

In first go brute force approach will be make a new linked list and maintain two pointer point to head of two linked list and compare then and insert into list

Code:

```
  ListNode *dummy, *temp;
        dummy = new ListNode();
        temp = dummy;
        //when both list1 and list2 isn't empty
        while(list1 && list2){
            if(list1->val < list2->val){
                temp->next = list1;
                list1 = list1->next;
            }
            else{
                temp->next = list2;
```

```
                list2 = list2->next;
            }
            temp = temp->next;
        }

        // we reached at the end of one of the list
        if(list1) temp->next = list1;
        if(list2) temp->next = list2;

        return dummy->next;
    }
```

Second approach will be insert into list 1 according to conditions

Code:

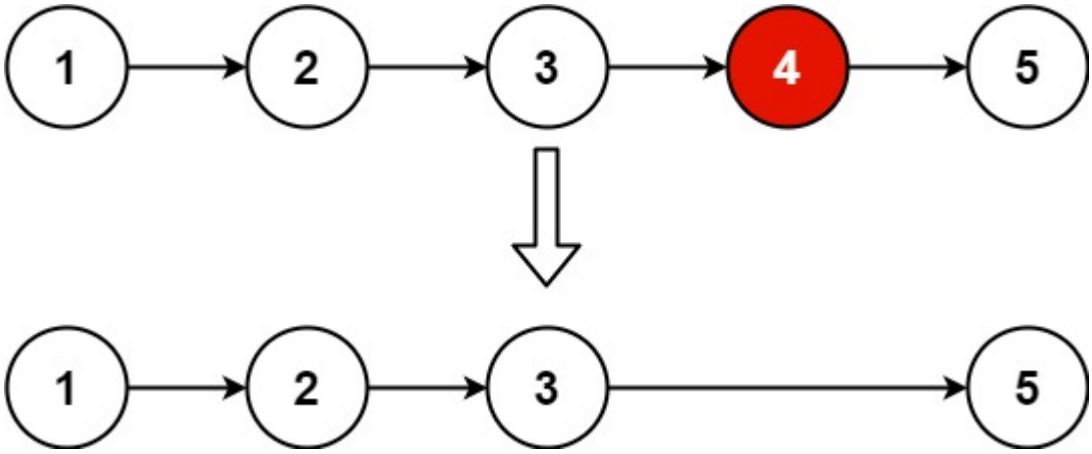
```
    ListNode* mergeTwoLists(ListNode* list1, ListNode* list2) {
        if(list1 == NULL)
            return list2;
        if(list2 == NULL)
            return list1;
        ListNode * ptr = list1;
        if(list1 -> val > list2 -> val)
        {
            ptr = list2;
            list2 = list2 -> next;
        }
        else
        {
            list1 = list1 -> next;
        }
        ListNode *curr = ptr;

        while(list1 &&  list2)
        {
```

```
            if(list1 -> val < list2 -> val){
                curr->next = list1;
                list1 = list1 -> next;
            }
            else{
                curr->next = list2;
                list2 = list2 -> next;
            }
            curr = curr -> next;

        }


        if(!list1)
            curr -> next = list2;
        else
            curr -> next = list1;

        return ptr;

    }
};
```

Remove Nth Node From End of List

Given the head of a linked list, remove the n^{th} node from the end of the list and return its head.

Ans:



Observation and approach : process based question

In first go brute force approach will be travel to give location and maintain two pointer which point one location behind and one location forward and delete the element and link the old one

Proper explanation:

The main goal of this approach is to use two pointers to iterate over the linked list and they must be seperated by a distance of n. So that when one of the pointers reach the end of the linked list, ther other will be pointing to the node just before the node that must be removed

1. Initialize two pointers, slow and fast, to the head of the linked list.
2. Move the fast pointer n positions ahead in the linked list. This will create a gap of n nodes between the fast and slow pointers.
3. If the fast pointer becomes NULL, it means n is equal to the length of the linked list. In this case, the first node (head) needs to be removed. Update the head pointer to the second node and delete the first node pointed to by slow. Return the updated head.
4. While the fast pointer and its next node are not NULL, move both the fast and slow pointers one step forward. This ensures that the gap between them remains n nodes.
5. Once the fast pointer reaches the end of the linked list, the slow pointer will be pointing to the node just before the node to be removed.
6. Update the next pointer of the slow node to skip the node to be removed and instead point to the node after it. This effectively removes the node from the linked list.
7. Delete the node that slow->next was pointing to.

8. Return the head of the updated linked list.

Code:

```
ListNode *slow=head, *fast=head;

        for(int i=0; i<n; i++)
            fast = fast->next;

        if(!fast)
        {
            head = head->next;
            delete slow;    // Since slow points to the first node
            return head;
        }

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

        if(slow && slow->next)
        {
            ListNode* temp = slow->next;
            slow->next = slow->next->next;
            delete temp;
        }

        return head;
```

Second approach will be separate the parts one part before deletion node one part after deletion node the just make a link between them

Code:

```
int Size(ListNode* head){
        int cnt = 0;
        while(head!=NULL){
            cnt++;
            head=head->next;
        }
        return cnt;
    }
    ListNode* removeNthFromEnd(ListNode* head, int n) {
        int m = Size(head);
        if(m == 1){
            head = NULL;
            return head;
        }
        if(m-n == 0){
            ListNode* cur = head;
            head=head->next;
            cur = NULL;
            return head;
        }
        ListNode* prev = NULL;
        ListNode* cur = head;
        for(int i = 0;i<(m-n);i++){
            prev = cur;
            cur = prev->next;
        }
        prev->next = cur->next;
        cur->next=NULL;
        delete cur;
        return head;  }
```

Add Two Numbers

You are given two non-empty linked lists representing two non-negative integers. The digits are stored in reverse order, and each of their nodes contains a single digit. Add the two numbers and return the sum as a linked list.

You may assume the two numbers do not contain any leading zero, except the number 0 itself.

Ans:

Observation and approach :process based

First approach is to convert thus linked list to number add then and again make a linked list

Second approach will be do this things on linked list only means to add

Then we have to Handel case for carry

Code:

```
void insertAtTail(ListNode *&head, ListNode *&tail, int d){
        ListNode *temp = new ListNode(d);
        if(head==NULL){
            head = temp;
            tail = temp;
        }
        else{
            tail->next = temp;
            tail = temp;
        }
        return;
    }
    ListNode *add(ListNode *a, ListNode *b){
        int carry = 0;
        ListNode *ansHead = NULL;
        ListNode *ansTail = NULL;

        while(a!=NULL || b!=NULL || carry!=0){
```

```
        int val1 = 0;
        if(a!=NULL) val1 = a->val;
        int val2 = 0;
        if(b!=NULL) val2 = b->val;
        int sum = carry + val1 + val2;
        int digit = sum%10;
        insertAtTail(ansHead, ansTail, digit);
        carry = sum/10;
       if(a!=NULL) a = a->next;
        if(b!=NULL) b = b->next;
    }
    return ansHead;
}

ListNode* addTwoNumbers(ListNode* l1, ListNode* l2) {
    ListNode *ans = add(l1,l2);
    return ans;
}
```

Delete Node in a Linked List

There is a singly-linked list head and we want to delete a node node in it.

You are given the node to be deleted node. You will not be given access to the first node of head.

All the values of the linked list are unique, and it is guaranteed that the given node node is not the last node in the linked list.

Delete the given node. Note that by deleting the node, we do not mean removing it from memory. We mean:

- The value of the given node should not exist in the linked list.
- The number of nodes in the linked list should decrease by one.
- All the values before node should be in the same order.
- All the values after node should be in the same order.

Ans:

Simple delete the node 😅

Code:

```
static void deleteNode(ListNode* node) {
        ListNode* next = node->next;
        *node = *next;
        delete next;
    }
```