### QEasy Level:

### Q1. Finding middle element in a linked list.

### Example:

### Odd length

### 1 2 3 4 5 output should be 3.

### Even length

### 1 2 3 4 5 6 output should be 4 not 3.

### Approach 1 is:

### We can take two pointers .

### Traversal the linklist

### Traversal the whole linklist .

### Fast pointer move 2 node and second pointer move 1 node.

### Even:

### 1 2 3 4 5 6 NULL

### F1 F2 F3 F4

### S1 S2 s3 S4

### Odd:

### 1 2 3 4 5

### F1 F2 F3

### S1 S2 s3

### Code:

### int getMiddle(Node \*head)

### {

### if(head==NULL)return NULL;

### Node \*slow=head,\*fast=head;

### while(fast!=NULL&&fast->next!=NULL)

### {

### slow=slow->next;

### fast=fast->next->next;

### }

### return slow->data;

### }

### 

### Q2. Reverse a linked list:

### Using three node pointer.

### First=head, second=NULL. Third=NULL

### Traversal the link list

### And update the node value like

### third=second,

### second=first,

### first=first->next.

### second->next=first;

### Return second.

### NL NL 1 2 3 4

### F1 f2 f3 f4 f5

### S1 s2 s3 s4 s5

### T1 T2 t3 t4 t5

### S2->t1 S2->t2 s3->t3 s4->t4 s5->t5

### Print : NL 1 2 3 4

### 

### Q3. Reverse a doubly linked list(Link List Notes).

### Using one extra node.

### Reverse the next to prev.

### While(curr!=NULL)

### Temp=curr->prev;

### Curr->prev=curr->next;

### Curr->next=temp;

### Curr=curr->prev;

### 4. Temp==NULL?head:temp->prev;

### Q4. Rotate a Linked List

### 

### Approach:

### N= 8

### 1 2 3 4 5 6 7 8 k= 4

### Step1: check the edge case that is head or head next is null return head. (in this either 0 or 1 node means any times of rotation values always same.)

### Step2: compute the length of link list as well as traversal node assign to last node.

### Like that:

### Len=1;

### while(last->next!=NULL)

### { len++;

### Last=last->next;

### }

### Hear, the last node assign the last node of the link list and also calculate the list of the node.

### Step3: check k==len?head

### Step4: update k =k%len . it may be value of k is greater the n.

### Step5: again traversal the list until temp!=NULL and k>1. In the case the temp node assign the node where we can break.

### Step6: compute the connection between the node.

### Step7: finally, return head.

### while(temp!=NULL&&k>1){

### k--;

### temp=temp->next;}

### if(k!=0){last->next=head;

### head=temp->next;

### temp->next=NULL;}

### return head;

### head=h, last=l, temp=t(break point)

### 1 2 3 4 5 6 7 8

### H1

### L1 l2 l3 l4 l5 l6 l7 l8(l8 next null)

### T1 t2 t3 t4

### K=4 3 2 1(not go the next node. E]we need this node)

### Iteration 1 2 3

### Q5. Nth node from end of linked list

### Approach:

### 2 3 4 5 6 7 8 9

### N=5

### Output: 5

### In single traversal solve using recursive function:

### Step1: check the base condition. If head==null return null and also initialise one static variable. I=0.

### Step2: call the function.

### Step3: while returning the function. Increase the value of I and if n==I print the data.

**void** printNthFromLast(**struct** Node\* head, **int** N)

{

**static** **int** i = 0;

**if** (head == NULL)

**return**;

    printNthFromLast(head->next, N);

**if** (++i == N)

        cout<<head->data;

}

# Q6. Delete last occurrence of an item from linked list

### Q7. Delete Middle of Linked List

Approach 1:

Step1: if only one node return null.

Step2: took three pointers slowprev fast slow. With the help find middle node of prev node.

Step3: remove the middle node.

Code:

if(head->next==NULL)return NULL;

Node \*third=NULL,\*slow=head,\*fast=head;

while(fast&&fast->next){

third=slow;

slow=slow->next;

fast=fast->next->next;}

third->next=slow->next;

return head;

in odd

1 2 3 4 5 6 7

F1 f2 f3 f4 connect t4 next to s4 next;

S1 s2 s3 s4

T1 t2 t3 t4

in even

1 2 3 4 5 6

F1 f2 f3 ->connect t3 next to s3 next;

S1 s2 s3

T1 t2 t3

### Q8. Remove duplicate element from sorted Linked List

### Approach:

### 

### Step1: consider two point. Assign the value temp1=head , temp2=head->next;

### Step2: traversal the list and check if temp1 data are same as temp2 data

### Remove the node temp1 and temp1 move next node,

### Temp2->next=temp1->next;

### Temp1=temp2->next;

### Step3: edge case if last node of data are equal then temp2->next=NULL;

### Step4: return head;

### Example

### 1 2 2 2 3 4 4 4

### S1 s2 s3 s4

### f1 f2 f2 f2 f3 f4 f4

### at last check s4==f4 then assign s4=NULL.

### Code:

### if(head==NULL||head->next==NULL)return head;

### Node \*temp1=head->next,\*temp2=head;

### while(temp1!=NULL){

### while(temp1->data==temp2->data&&temp1->next){

### temp2->next=temp1->next;

### temp1=temp2->next;}

### if(temp1->data==temp2->data){

### temp2->next=NULL;}

### temp2=temp1;

### temp1=temp1->next;}

### return head;

### Q9. Detect Loop in linked list

## Approach1:Using [**Floyd’s Cycle-Finding Algorithm**](https://www.geeksforgeeks.org/floyds-cycle-finding-algorithm/)**:**

### Consider two pointers slow and fast, slow move to next step and fast move to 2 steps.

### If any point slow node equal to fast return 1;

### Code: Node \*slow=head,\*fast=head;

### while(slow&&fast&&fast->next!=NULL){

### slow=slow->next;

### fast=fast->next->next;

### if(slow==fast)return 1;}// last line compare.

### return false;

### Approach2:using :

## **Detect loop in a linked list by Modification In Node Structure:**

*The idea is to modify the node structure by adding****flag****in it and mark the flag whenever visit the node.*

Follow the steps below to solve the problem:

* Have a visited flag with each node.
* Traverse the linked list and keep marking visited nodes.
* If you see a visited node again then there is a loop.

Struct Node{

Int data;

Node \* next;

Int flag;

};

Approach3:Using HashTable:

T*he idea is to insert the nodes in the****hashmap****and whenever a node is encountered that is already present in the hashmap then return true.*

### Q10. Delete N nodes after M nodes of a linked list

### Given a linked list, delete N nodes after skipping M nodes of a linked list until the last of the linked list.

### Approach1:

### M= 2, N=2

### 1 2 3 4 5 6 7 8 9

### T1 t2

### F s s=M+N;

### Count: 1 2 3 4 5 6 7 8 9

### Assign first->next=second->next;

### Code:

### Node \*temp=head,\*first=NULL;

### int count=0;

### while(temp!=NULL){

### for(count=1;count<M&&temp!=NULL;count++) {

### temp=temp->next;}

### if(temp==NULL)return;

### // it may possible that not next n element is present

### first=temp->next;

### for(count=1;count<=N&&first!=NULL;count++){

### first=first->next;}

### temp->next=first;

### temp=first;}

### Q11. Merge Lists Alternatingly

### 1 2 3 4 5

### 6 8 9

### Using three pointer connect allthe node.

### void mergeList(struct Node \*\*p, struct Node \*\*q)

### {

### // Code here

### if(\*p==NULL||\*q==NULL)return;

### struct Node \*head=\*p;

### while(\*p&&\*q)

### {

### struct Node \*temp1=(\*p)->next;

### struct Node \*temp2=\*q;

### (\*q)=(\*q)->next;

### (\*p)->next=temp2;

### temp2->next=temp1;

### \*p=temp1;

### }

### \*p=head;

### }

### Q12. Circular Linked List Traversal

### Approach:

### First print the first node and traversal the temp assign temp next.

### While(temp!=head ) print all the node data.

### Approach 2: using do while loop but one edge case is check the initial node is not null.

### Q13.Deletion of the circular list and also reverse it.

### Delete the circular Node :

### If head is null return;

### If only one node assign head=null

### If delete first node then traversal all the node at the last node and connect last next to first next and assign head as first next node that is second node.

### If any not in between the last and first then traversal the with two pointer

### Assign temp2 next to temp1 next;

### If last node delete the assign second last to first node return head.

### Code: <https://www.geeksforgeeks.org/problems/deletion-and-reverse-in-linked-list/1?itm_source=geeksforgeeks&itm_medium=article&itm_campaign=>

### Reverse the circular Linklist:

### Traversal with the three pointers. similar to single link list.

### And last some change the node connection.

### Head->next=second;

### Head=second; until first->next!=NULL; or return head;

### Q14). Delete without head pointer

### LINK: <https://www.geeksforgeeks.org/problems/delete-without-head-pointer/1?itm_source=geeksforgeeks&itm_medium=article&itm_campaign=bottom_sticky_on_article>

**Input:**

N = 4

value[] = {10,20,4,30}

node = 20

**Output:** 10 4 30

**Explanation:** After deleting 20 from

the linked list, we have remaining

nodes as 10, 4 and 30.

### Approach:

### Check the given node is null or its last node(node->next==null return)

### Else del->data=del->next->data(data will assign next data value)

### And address will assign node of next node.

### Del->next=del->next->next;