SNo.	Problem Statement

1.	**Easy LeveL : Middle of the Linked List.**
	Code:
	
	Input: head = [1,2,3,4,5]
	
	Output: [3,4,5]
	
	Explanation: The middle node of the list is node 3.
	
	ListNode* middle(ListNode* head)
	{
	ListNode* slow=head;
	ListNode* fast=head;
	if(head!=NULL)
	while(fast!=NULL and fast->next!=NULL)
	{
	fast=fast->next->next;
	slow=slow->next;
	}
	return slow;
	}

2.	**Easy Level : Linked List Cycle**
	Code:
	
	Input: head = [3,2,0,-4], pos = 1
	
	Output: true
	
	Explanation: There is a cycle in the linked list, where the tail connects to the 1st node (0-indexed).
	
	bool hasCycle(ListNode *head) {
	
	ListNode*slow=head;
	ListNode*fast=head;
	while(fast!=NULL && fast->next!=NULL){
	slow=slow->next;
	fast=fast->next->next;

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

3. Easy Level : Convert Binary Number in a Linked List to Integer.

Code:

```
Input: head = [1,0,1]

Output: 5

Explanation: (101) in base 2 = (5) in base 10
```

```
int getDecimalValue(ListNode* head) {

    int num=head->val;
    while(head->next!=NULL)
    {
        num=num*2+head->next->val;
        head=head->next;
    }
    return num;
}
```

4. Easy Level : Remove Duplicates from Sorted List.

Code:

```
Input: head = [1,1,2]

Output: [1,2]
```

```
ListNode* removeduplicate(ListNode* head){

    if(head==NULL)
    return head;
    ListNode* tmp=head;
    while(tmp->next!=NULL)
    {
        if(tmp->next->val==tmp->next->val)
        tmp->next=tmp->next->next;
```

```
        else
        tmp=tmp->next;
    }
    return head;
}
```

5. **Easy Level : Sort a linked list of 0s, 1s and 2s.**

Code:

Input: 1 -> 1 -> 2 -> 0 -> 2 -> 0 -> 1 -> NULL
Output: 0 -> 0 -> 1 -> 1 -> 1 -> 2 -> 2 -> NULL
Input: 1 -> 1 -> 2 -> 1 -> 0 -> NULL
Output: 0 -> 1 -> 1 -> 1 -> 2 -> NULL

```
ListNode* sortList(ListNode* head)
{
    vector<int>v;
    if(head==NULL || head->next==NULL)
    return head;
    while(head!=NULL)
    {
        v.push_back(head->val);
        head=head->next;
    }
    sort(v.begin(),v.end());
    ListNode* node=new ListNode(v[0]);
    ListNode* start=node;
    for(int i=1;i<v.size();i++)
    {
        node->next=new ListNode(v[i]);
        node=node->next;
    }
    return start;
}
```

6. **Easy Level : Remove Linked List Elements.**

Code:

```
Input: head = [1,2,6,3,4,5,6], val = 6

Output: [1,2,3,4,5]
```

```
ListNode* removeElements(ListNode* head, int val) {
```

```
        if(head==NULL)
            return NULL;
        head->next=removeElements(head->next,val);
        if(head->val==val)
            return head->next;
        return head;
    }
```

7. **Easy Level : Merge Two Sorted Lists.**

Code:

```
Input: list1 = [1,2,4], list2 = [1,3,4]

Output: [1,1,2,3,4,4]
```

```
ListNode* mergeTwoLists(ListNode* l1, ListNode* l2) {

        ListNode *ans=NULL;
        if(!l1)
            return l2;
        else if(!l2)
            return l1;
        if(l1->val <= l2->val)
        {
            ans=l1;
            ans->next=mergeTwoLists(l1->next,l2);
        }
        else
        {
            ans=l2;
            ans->next=mergeTwoLists(l1,l2->next);
        }
        return ans;
    }
```

8. **Easy Level : Multiply two numbers represented by Linked Lists.**

Code:

```
Input : 9->4->6
        8->4
Output : 79464
```

```
Input : 3->2->1
        1->2
Output : 3852
```

```
long long multiplyTwoLists (Node* l1, Node* l2)
{
   long long N= 1000000007;
   long long num1 = 0, num2 = 0;
   while (l1 || l2){

      if(l1){
         num1 = ((num1)*10)%N + l1->data;
         l1 = l1->next;
      }

      if(l2)
      {
         num2 = ((num2)*10)%N + l2->data;
         l2 = l2->next;
      }

   }
   return ((num1%N)*(num2%N))%N;
}
```

9. **Easy Level : Intersection of Two Linked Lists.**

Code:

```
Input: intersectVal = 8, listA = [4,1,8,4,5], listB = [5,6,1,8,4,5],
skipA = 2, skipB = 3

Output: Intersected at '8'
```

```
ListNode *getIntersectionNode(ListNode *headA, ListNode *headB)
{


   if(headA == NULL || headB == NULL)
   return NULL;
```

```
    ListNode* a=headA;
    ListNode* b=headB;
    while(a!=b)
    {
        a = a == NULL? headB : a->next;
        b = b == NULL ? headA : b->next;
    }
    return a;
}
```

10. **Easy Level : Given only a pointer/reference to a node to be deleted in a singly linked list, how do you delete it?**

Code:

```
void deleteNode(Node* node)
{
    Node* prev;
    if(prev==NULL)
      return;
    else
    {
       while(node->next!=NULL)
       {
          node->data=node->next->data;
          prev=node;
          node=node->next;
       }
       prev->next=NULL;
    }
}
```

11. **Easy Level : Palindrome Linked List.**

Code:

```
Input: head = [1,2,2,1]

Output: true
```

```
bool isPalindrome(ListNode* head)
{
    stack<int>s;
    ListNode* slow=head;
    ListNode* fast=head;
```

```
    while(fast and fast->next)
    {
       s.push(slow->data);
       slow=slow->next;
       fast=fast->next->next;

    }
    if(fast!=NULL)
     slow=slow->next;
      while(!s.empty() and slow)
       {
          if(s.top()!=slow->val)
             return false;
          s.pop();
          slow=slow->next;
       }
    return true;
}
```

12. **Easy Level : Reverse Linked List.**

Code:

```
Input: head = [1,2,3,4,5]

Output: [5,4,3,2,1]
```

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