EASY :

1. 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(fast==slow)
            return true;
        }
        return false;
    }
};
```

2. Reverse linked list

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

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

        ListNode* rest=reverseList(head->next);
        head->next->next=head;
        head->next=NULL;
        return rest;
    }
};

// second approach

class Solution {
public:
    ListNode* reverseList(ListNode* head)
    {
        ListNode *temp;
        ListNode *prev=0;
        while(head!=NULL)
        {
            temp = head->next;
            head->next = prev;
            prev = head;
            head = temp;
        }
        return prev;
     }
};
```

3. Middle of the linked link

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

        ListNode *temp;
        ListNode *prev=0;
        while(head!=NULL)
        {
            temp = head->next;
            head->next = prev;
            prev = head;
            head = temp;
        }
        return prev;

    }
};
```

4. Remove duplicates from linked list

```
class Solution {
public:
    ListNode* deleteDuplicates(ListNode* head)
    {
      ListNode* temp=head;
      while(temp)
      {
        ListNode* tempp=temp->next;
        while(tempp && temp->val==tempp->val)
        tempp=tempp->next;

        temp->next=tempp;
        temp=temp->next;
    }
    return head;
    }
};
```

5. Maximum twin sum linked list

```
class Solution {
public:
    int pairSum(ListNode* head)
    {
        if(!head)
        return 0;
        vector<int>v;
        while(head)
        {
            v.push_back(head->val);
            head=head->next;

        }
        int n=v.size();
        int maxx=0;
        for(int i=0;i<n/2;i++)
        maxx=max(maxx,v[i]+v[n-i-1]);
        return maxx;
     }
};
```

6. Intersection point of linked lists

```
class Solution {
public:
    ListNode *getIntersectionNode(ListNode *headA, ListNode *headB)
    {
       if(headA == headB)
        return headA;

        ListNode *temp1 = headA;
        ListNode *temp2 = headB;
        while(temp1 != temp2)
        {
           temp1 = (temp1 == NULL ? headB : temp1->next);
           temp2 = (temp2 == NULL ? headA : temp2->next);
        }
        return !temp1 ? NULL : temp1;
    }
};
```

MEDIUM :

1. Delete loop in linked list

```
void removeLoop(ListNode* head) {
        if(!head)
        return ;
        ListNode* fast = head;
        ListNode *slow = head;
        int f=0;
        while(fast->next and fast->next->next)
        {
            fast = fast->next->next;
            slow = slow->next;
            if(fast == slow)
            {

                slow = head;
                while(fast != slow)
                {
                    fast = fast->next;
                    slow = slow->next;
                }
                f=1;
            }
            if(f==1)
            break;
        }
         if(f==1)
         {
        while(slow->next!=fast)
        slow=slow->next;
        slow->next=0;
         }
}
```

2. Swapping node in linked list

```
class Solution {
public:
    ListNode* swapNodes(ListNode* head, int k)
    {
         if(!head)
         return 0;

        ListNode* temp=head;
        int c=1;
        while(temp->next)
        {
            temp=temp->next;
            c++;
        }
        if(k==1 || k==c)
        {
            int f=head->val;
            head->val=temp->val;
            temp->val=f;
            return head;
        }
        if(c>k)
        {
            temp=head;
            int s=k;
            k--;
            while(k-- && temp)
            temp=temp->next;
            ListNode* tmp=head;
            c=c-s;
            while(c-- && tmp)
            tmp=tmp->next;
             c=tmp->val;
            tmp->val=temp->val;
            temp->val=c;
        }
        return head;
    }
};
```

3. Flatten a multilevel doubly linked list

```
class Solution {
public:
    Node* flatten(Node* head)
    {
        Node* cur=head;

        while(cur && !cur->child)
        cur=cur->next;
        if(!cur)
        return head;

        Node* temp=flatten(cur->next);
        Node* tmp=flatten(cur->child);
        cur->child = 0;
        cur->next = tmp;
        tmp->prev = cur;
        while(cur->next)
        cur = cur->next;
        cur->next = temp;
        if(temp)
        temp->prev = cur;
        return head;
    }
};
```

4. Add two number represented by linked list

```
class Solution {
public:
    ListNode* addTwoNumbers(ListNode* l1, ListNode* l2)
    {
        stack<int>st1;
        stack<int>st2;
        int sum=0;
        ListNode* ans=new ListNode(0);

        while(l1)
        {
            st1.push(l1->val);
            l1=l1->next;
        }
        while(l2)
        {
            st2.push(l2->val);
            l2=l2->next;
        }
        while(!st1.empty() || !st2.empty())
        {
            if(!st1.empty())
            {
                sum+=st1.top();
                st1.pop();
            }
            if(!st2.empty())
            {
                sum+=st2.top();
                st2.pop();
            }

            ans->val=sum%10;
            sum/=10;
            ListNode * head=new ListNode(sum);
            head->next=ans;
            ans=head;
        }
        return ans->val==0?ans->next:ans;
    }
};
```

HARD :

1. Merge k sorted linked list

```
class Solution {
public:
    ListNode* mergeKLists(vector<ListNode*>& lists)
    {

        priority_queue<pair<int,int>,vector<pair<int,int>>,greater<pair<int,int>>>pq;
        for(int i=0;i<lists.size();i++)
        {
            if(lists[i])
            pq.push({lists[i]->val,i});
         }

        ListNode* ans=new ListNode(-1);
        ListNode* temp=ans;
         while(!pq.empty())
        {
            int v=pq.top().first;
            int index=pq.top().second;
            pq.pop();
            temp->next=new ListNode(v);
            lists[index]=lists[index]->next;

            if(lists[index])
            pq.push({lists[index]->val,index});
            temp=temp->next;
         }
        return ans->next;
    }
};
```