Linked List

1.Pallindrome Linked List

Given a singly linked list, determine if its a palindrome. Return 1 or 0 denoting if its a palindrome or not, respectively.
Notes:

- Expected solution is linear in time and constant in space.

For example,
List 1-->2-->1 is a palindrome.
List 1-->2-->3 is not a palindrome.

Solution:

```
int Solution::lPalin(ListNode* A) {
    vector<int> v;
    ListNode *temp=A;

    while(temp!=NULL){
        v.push_back(temp->val);
        temp=temp->next;
    }
    int i=0;
    int j=v.size()-1;
    while(i<j){
      if(v[i]!=v[j]){
          return 0;
      }
       i++;
       j--;
    }
    return 1;


}
```

Second Approach

```
      ListNode *reverse(ListNode *A){
          ListNode *root = new ListNode(-1);
```

```
    root->next = A;
    ListNode *curr = root->next, *prev = root;
    while(curr){
        ListNode * next = curr->next;
        curr->next = prev;
        prev = curr;
        curr = next;
    }
    ListNode* next = root->next;
    next->next = NULL;
    root->next = NULL; ////

    return prev;
}

int Solution::lPalin(ListNode* A){

    ListNode *slow = A, *fast = A, *prev = NULL;
    ListNode *root = new ListNode(-1);
    root->next = A;
    prev = root;
    while(slow && fast && fast->next){
        slow = slow->next;
        prev = prev->next;
        fast = fast->next->next;
    }
    /// two case
    auto pr = reverse(prev->next);
    prev->next = NULL;

    auto right = pr;
    auto left = A;
    // print(left);
    // print(right);
    while(left && right){
        if(left->val != right->val) break;
        left = left->next;
        right = right->next;
    }
    if ((!left && !right) || (!left && right && !right->next)) return true;

    return false;
}
```

JAVA

In java we don’t have pointers so what approach we have is

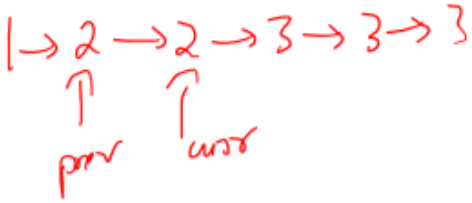
```
public class Solution {
    public int lPalin(ListNode A) {
        StringBuilder s1 = new StringBuilder();
        StringBuilder s2 = new StringBuilder();

        if(A==null||A.next==null)return 1;
        ListNode p=null, c=A , n = A.next;
        while(n!=null){
            s1.append(c.val);
            c.next = p;
            p = c;
            c = n;
            n=n.next;
        }
        s1.append(c.val);
        c.next = p;
        while(c!=null){
            s2.append(c.val);
            c = c.next;
        }

        return s1.toString().equals(s2.toString())?1:0;
    }
}
```

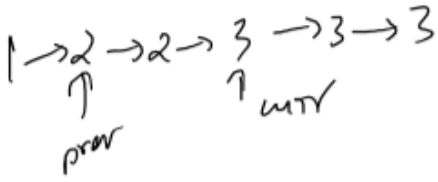
2.Remove Duplicate In List I

Given a sorted linked list, delete all duplicates such that each element appear only once.
For example,
Given 1->1->2, return 1->2.
Given 1->1->2->3->3, return 1->2->3.

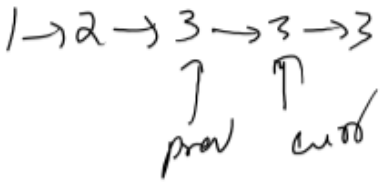


now as values are same we increment curr till prev->value == curr->value

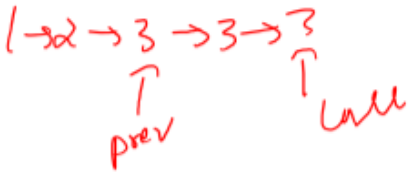
so



now, prev->next = curr & increment both pointers

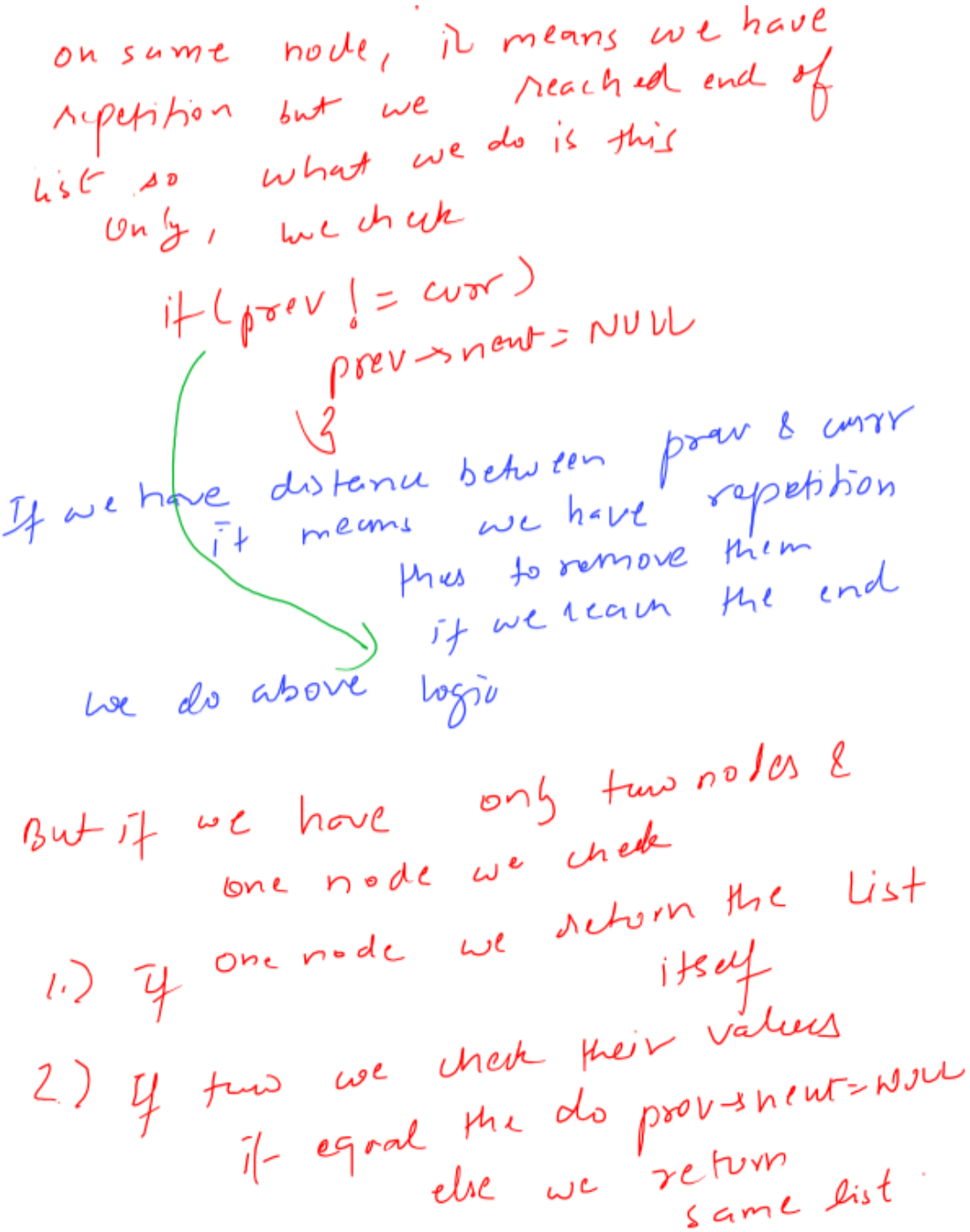


=> prev->value == curr->value so we increment



same but we reach the end

we can see prev & curr are not



Solution:

```
ListNode* Solution::deleteDuplicates(ListNode* A) {

    ListNode *temp=A;
   ListNode *prev=A;
   temp=temp->next;
    while(temp!=NULL){
        if(temp->val!=prev->val){
            prev->next=temp;
            prev=temp;
```

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

    if(prev!=temp){
        prev->next=NULL;
    }

    return A;
}
```

JAVA

```
public class Solution {
    public ListNode deleteDuplicates(ListNode A) {
         ListNode temp=A;
   ListNode prev=A;
   temp=temp.next;
    while(temp!=null){
        if(temp.val!=prev.val){
            prev.next=temp;
            prev=temp;
        }
        temp=temp.next;
    }

    if(prev!=temp){
        prev.next=null;
    }

    return A;
    }
}
```

3. Remove Duplicates In List II(Same as above with variation)

```
ListNode* Solution::deleteDuplicates(ListNode* A) {
    ListNode *rt=NULL;
    ListNode *rttemp=rt;
    ListNode* temp=A;

    map<int,int> mp;
    if(A->next==NULL){
        rt=new ListNode(A->val);
        return rt;
```

```
    }

    while(temp!=NULL){
      mp[temp->val]++;
       temp=temp->next;
    }

    int flag=0;

    for(map<int,int>::iterator it=mp.begin();it!=mp.end();it++){
        if(it->second==1){
            if(flag==0){
                rt=new ListNode(it->first);
                rttemp=rt;
                flag=1;

            }else{
            rttemp->next=new ListNode(it->first);
            rttemp=rttemp->next;
    }

        }
    }


    return rt;
}
```

4.Merge Two Sorted List

Merge two sorted linked lists and return it as a new list.
The new list should be made by splicing together the nodes of the first two lists, and should also be sorted.
For example, given following linked lists :

```
 5 -> 8 -> 20
 4 -> 11 -> 15
```

The merged list should be :

```
4 -> 5 -> 8 -> 11 -> 15 -> 20
```

Solution:

```
ListNode* Solution::mergeTwoLists(ListNode* A, ListNode* B) {
    ListNode *head=NULL;
```

```
    ListNode *comb=NULL;

    if(A->val<B->val){
        head=new ListNode(A->val);
        comb=head;
         A=A->next;
    }else{
           head=new ListNode(B->val);
           comb=head;
         B=B->next;
    }

     while(A!=NULL && B!=NULL){
         if(A->val<B->val){
            comb->next=new ListNode(A->val);
            comb=comb->next;
            A=A->next;
         }else {
              comb->next=new ListNode(B->val);
            comb=comb->next;
            B=B->next;
         }
     }


     if(A!=NULL){
          comb->next=A;
     }

     if(B!=NULL){
           comb->next=B;
     }
    return head;
}
```

JAVA:

```
public class Solution {
    public ListNode mergeTwoLists(ListNode A, ListNode B) {
         ListNode head=null;
    ListNode comb=null;

    if(A.val<B.val){
        head=new ListNode(A.val);
        comb=head;
         A=A.next;
```

```
    }else{
          head=new ListNode(B.val);
          comb=head;
         B=B.next;
    }

     while(A!=null && B!=null){
         if(A.val<B.val){
            comb.next=new ListNode(A.val);
            comb=comb.next;
            A=A.next;
         }else {
              comb.next=new ListNode(B.val);
            comb=comb.next;
            B=B.next;
         }
     }


     if(A!=null){
          comb.next=A;
     }

     if(B!=null){
          comb.next=B;
     }
    return head;
    }
}
```

5.Remove Nth Node from list

Given a linked list, remove the nth node from the end of list and return its head.
For example,
Given linked list: 1->2->3->4->5, and n = 2.
After removing the second node from the end, the linked list becomes 1->2->3->5.

Note:

- If n is greater than the size of the list, remove the first node of the list.

Try doing it using constant additional space.

```
/**
 * Definition for singly-linked list.
 * struct ListNode {
```

```
 *     int val;
 *     ListNode *next;
 *     ListNode(int x) : val(x), next(NULL) {}
 * };
 */



ListNode* Solution::removeNthFromEnd(ListNode* A, int B) {

    ListNode *temp = A;
    int size =0;
    while(temp!=NULL){
        size++;
        temp=temp->next;
    }

    if(size==1){
        return NULL;
    }

    if(size<B || size==B){
        A=A->next;
        return A;
    }else{

        int index=size-1-B;
        int ci=0;
        ListNode *temp=A;
        while(ci<index){
            temp=temp->next;
            ci++;
        }

        temp->next=(temp->next)->next;
    }

    return A;

}
```

6. Reverse Linked List(Between A and C)

```
      ListNode* Solution::reverseBetween(ListNode* A, int B, int C) {

          int i=1;
```

```
        ListNode *start, *end, *prev, *next;

        ListNode *temp=new ListNode(0);

        temp->next=A;

        ListNode *curr=temp;

        while(i != (B)){
           curr=curr->next;
           i++;
        }


        start=curr;
        end=curr->next;
        curr=curr->next;
        prev=NULL;

        while(i != (C+1)){
           next=curr->next;
           curr->next=prev;

           prev=curr;
           curr=next;

           i++;
        }

        start->next=prev;
        end->next=curr;

        return temp->next;

    }
```

7. Reorder List

Given a singly linked list
L: L0 → L1 → … → Ln-1 → Ln,
reorder it to:
L0 → Ln → L1 → Ln-1 → L2 → Ln-2 → …
You must do this in-place without altering the nodes' values.
For example,
Given {1,2,3,4}, reorder it to {1,4,2,3}.

Solution:

```
public class Solution {
    public  ListNode reverseList(ListNode root) {
        ListNode prev = null;

        ListNode curr = root;

        ListNode next = null;

        while (curr != null) {
            // System.out.println(curr.val);
            next = curr.next;
            curr.next = prev;
            prev = curr;
            curr = next;
        }

        return prev;
 }
    public ListNode reorderList(ListNode A) {


        if(A == null) return A;

        ListNode root = A;
        ListNode slow = root;
        ListNode fast = slow.next;
        while(fast!= null && fast.next != null) {
            slow = slow.next;
            fast = fast.next.next;
        }

        ListNode middleNode = slow.next;
        slow.next = null;

        ListNode r = reverseList(middleNode);

        ListNode temp = root;

        while(temp!= null && r!= null) {
            ListNode next1 = temp.next;
            temp.next = r;
            ListNode rNext = r.next;
            r.next = next1;
```

```
                r = rNext;
                temp = next1;
            }

            return root;


        }
    }
```

8. Add two numbers in Linked List

```
    ListNode *addTwoNumbers(ListNode *l1, ListNode *l2) {
            if(!l1)
                return l2;
            if(!l2)
                return l1;

            int carry = (l1->val + l2->val) / 10;
            ListNode *l3 = new ListNode((l1->val + l2->val) % 10);
            ListNode *tail = l3;
            l1 = l1->next;
            l2 = l2->next;

            while(l1 || l2 || carry)
            {
                int sum = ((l1 ? l1->val : 0) + (l2 ? l2->val : 0) + carry);
                ListNode *t  = new ListNode(sum % 10);
                carry = sum / 10;

                if(l1)
                    l1 = l1->next;
                if(l2)
                    l2 = l2->next;
                tail->next = t;
                tail = t;
            }

            return l3;
        }
    };
```

9. List Cycle

```
/**
```

```
 * Definition for singly-linked list.
 * struct ListNode {
 *     int val;
 *     ListNode *next;
 *     ListNode(int x) : val(x), next(NULL) {}
 * };
 */
ListNode* Solution::detectCycle(ListNode* A) {
    // Do not write main() function.
    // Do not read input, instead use the arguments to the function.
    // Do not print the output, instead return values as specified
    // Still have a doubt. Checkout www.interviewbit.com/pages/sample_codes/
for more details



    ListNode* first=A;
    ListNode *second=A;
    bool isCycle=false;

    while(first!=NULL && second!=NULL){
        first=first->next;
        if(second->next!=NULL){
        second=second->next->next;
        }else{
            return NULL;
        }

        if(first==second){
            isCycle=true;
            break;
        }
    }

    first=A;
    if(isCycle){
          while(first!=second){
      first=  first->next;
    second=  second->next;
    }
    }else{
        return NULL;
    }



    return first;
```

}

10. Swap Node List In Pairs

Given a linked list, swap every two adjacent nodes and return its head.
For example,
Given 1->2->3->4, you should return the list as 2->1->4->3.
Your algorithm should use only constant space. You may not modify the values in the list, only nodes itself can be changed.

```
ListNode* Solution::swapPairs(ListNode* A) {

   ListNode *first=A;

   while(first!=NULL &&  first->next!=NULL){
      int nxt=first->next->val;
       (first->next)->val=first->val;
       first->val=nxt;
       first=first->next->next;
   }

   return A;
}
```

11. Reverse K Linked List

```
ListNode* Solution::reverseList(ListNode* A, int B) {
    ListNode *curr = A, *prev = NULL, *next = NULL;
    int cnt =0;
     while(cnt<B && curr) {
        next = curr->next;
        curr->next = prev;
        prev = curr;
        curr = next;
        cnt++;
     }
     if(next) {
        A->next = reverseList(next, B);
     }
     return prev;
}
```

12.Rotate List

Given a list, rotate the list to the right by k places, where k is non-negative.
For example:
Given 1->2->3->4->5->NULL and k = 2,
return 4->5->1->2->3->NULL.

```
ListNode* Solution::rotateRight(ListNode* A, int B) {
  ListNode *temp=A;
  int count=1;
  while(temp->next!=NULL){
     count++;
     temp=temp->next;
  }

  int k=B%count;
  temp->next=A;


  for(int i=0;i<count-k;i++){
     temp=temp->next;
  }

  ListNode *head=temp->next;
  temp->next=NULL;

  return head;


}
```

13. **Partition List**

Given a linked list and a value x, partition it such that all nodes less than x come before nodes greater than or equal to x.
You should preserve the original relative order of the nodes in each of the two partitions.
For example,
Given 1->4->3->2->5->2 and x = 3,
return 1->2->2->4->3->5.

Code:

```
public class Solution {
   public ListNode partition(ListNode A, int B) {
      ListNode less=new ListNode(0);
      ListNode ls=less;
      ListNode more=new ListNode(0);
```

```
        ListNode mr=more;

        while(A!=null){
            if(A.val<B){
                ls.next=A;
                ls=ls.next;
            }else{
                mr.next=A;
                mr=mr.next;
            }

            A=A.next;
        }


        ls.next=more.next;
        mr.next=null;
        return less.next;


    }
}
```