Reverse Nodes in k groups

Count the length of linked list to find how many groups needs to be reversed by dividing cnt /k. Run a loop until cnt≥k to perform reverse and keep decrementing count by k once a group of k is reversed.

For reversing we take into account 3 pointers prev, nex, curr. We know that the first elem of the group to be reversed should always point to the next reversed group so always keep curr pointer at the first element. Create a dummy node for initial group and prev, curr and next all point to this dummy. now we know first elem of every group must be pointed by curr and curr will point to the next element of the last element of the previous group meaning if k=3 in list 1—2—3—4 then at the end of reversing curr→next will point to 4 because it is the next elem of the reversed group. so inside outer loop we write curr=prev→next. and next will point to current elem next which is next=curr→next

Now in 1—2—3—4 the link to be broken is 1—2 we know 1 next must point to starting of another group. so now curr is pointing to 1 so we shift 1 by one the curr→next because curr→next is pointing to the next elem whose link needs to be changes so moving step by step curr→next will point to next→next because that is the only elem the farthest we can move using these pointers.

now once curr→next =nex→next now 2—3 link needs to be broken so nex→next should now point to prev→next. now moving pointers prev→next=nex because we need to invert 3—4 link for 3, the prev element must be 2 which is pointed by nex now moving nex which will be nex=curr→next mpving nex by 1 position.

We know prev→next and dummy→next points to same elem so when we are changing prev→ next dummy→next also changes. But after one iteration we change prev=curr→next so now dummy is no more affected by prev because now prev is pointed to other elem.

Now for the next group we need to again set prev to curr→next because it was the last elem of previous group as we know that prev must point to the elem just before whose link needs to be altered. We also reduce cnt-=k;

At last dummy stores the head so we return dummy→next

```
class Solution {
public:
    ListNode* reverseKGroup(ListNode* head, int k) {
        int cnt=0;
        ListNode* temp=head;
        ListNode* dummy=new ListNode();
        dummy->next=head;
        ListNode *prev=dummy,*nex=dummy,*curr=dummy;
        while(temp!=NULL)
        {
            cnt++;
            temp=temp->next;
        }
        while(cnt>=k)
        {
            curr=prev->next;
            nex=curr->next;
            for(int i=1;i<k;i++)
            {
                curr->next=nex->next;
                nex->next=prev->next;
                prev->next=nex;
                nex=curr->next;
            }
            prev=curr;
            cnt-=k;
        }
        return dummy->next;
    }
};
```

- Time Complexity : O(N/K)*K which is equal to O(N)
- Space Complexity : O(1)