Reverse Linked List in groups of Size K

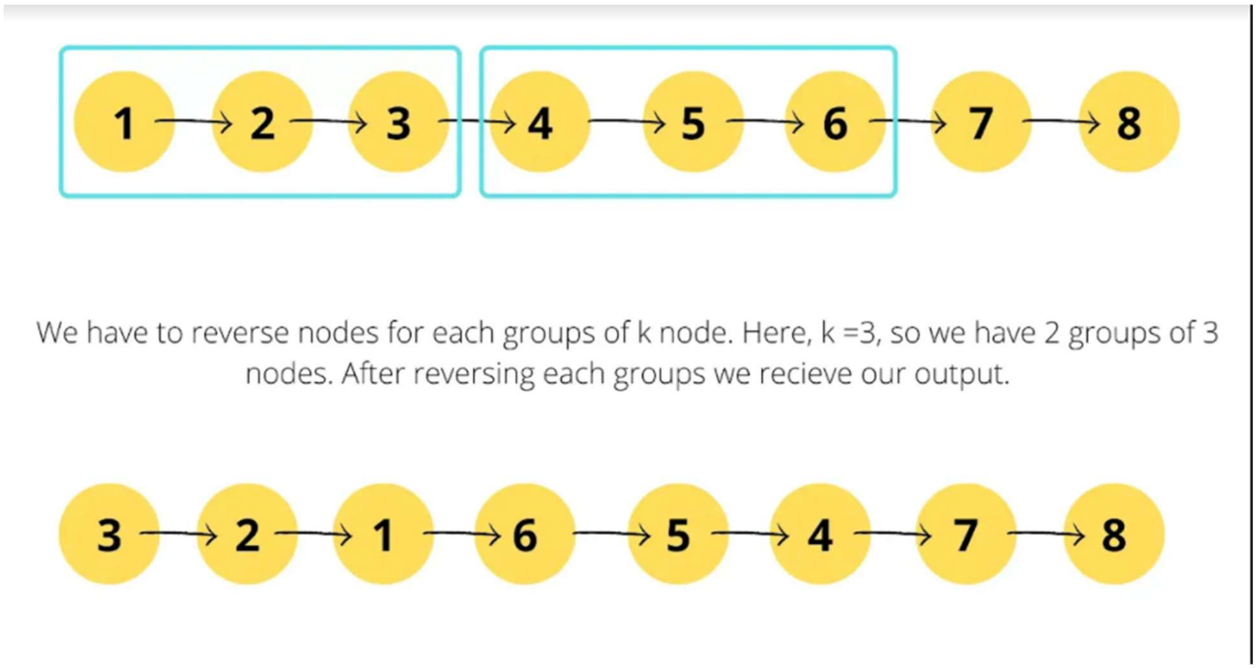
In this article, we will solve a very popular question Reverse Linked List in groups of Size K.

Problem Statement: Given the head of a linked list, reverse the nodes of the list k at a time, and return *the modified list*. k is a positive integer and is less than or equal to the length of the linked list. If the number of nodes is not a multiple of k then left-out nodes, in the end, should remain as it is.

Note: Also check this article to **reverse a linked list**

Examples:

```
Example 1:

Input:

 head = [1,2,3,4,5,6,7,8] k=3

Output:

 head = [3,2,1,6,5,4,7,8]

Explanation:
```



We have to reverse nodes for each groups of k node. Here, k =3, so we have 2 groups of 3 nodes. After reversing each groups we recieve our output.

Example 2:

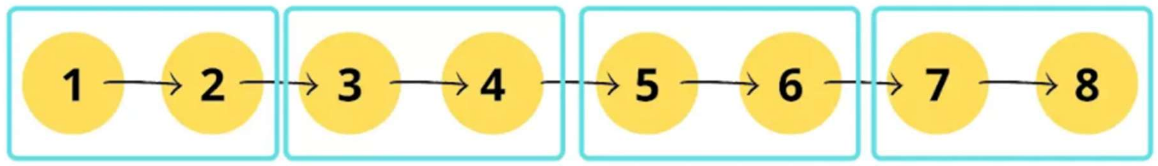
Input:

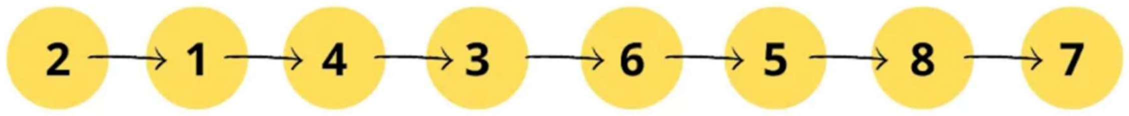
```
 head = [1,2,3,4,5,6,7,8] k=2
```

Output:

```
 head = [2,1,4,3,6,5,8,7]
```

Explanation:





We have to reverse nodes for each groups of k node. Here, k =2, so we have 3 groups of 2 nodes. After reversing each groups we recieve our output.

Solution:

Approach:

The following steps are needed to arrive at the desired output.

- Create a dummy node. Point next to this node to head of the linked list provided.
- Iterate through the given linked list to get the length of the list.
- Create three pointer pre,cur and nex to reverse each group. Why? If we observe output, we can see that we have to reverse each group, apart from modifying links of group.
- Iterate through the linked list until the length of list to be processed is greater than and equal to given k.
- For each iteration, point cur to pre->next and nex to nex->next.
- Start nested iteration for length of k.
- For each iteration, modify links as following steps:-

- cur->next = nex->next
- nex->next = pre->next
- pre->next = nex
- nex = cur->next

- Move pre to cur and reduce length by k.

Dry Run:

Code:

- C++ Code
- Java Code
- Python Code

```
#include<bits/stdc++.h>
using namespace std;

class node {
    public:
        int num;

        node* next;

        node(int a) {

            num = a;

            next = NULL;

        }

};

//utility function to insert node in the list
void insertNode(node* &head,int val) {
    node* newNode = new node(val);
    if(head == NULL) {
        head = newNode;
        return;
    }

    node* temp = head;
    while(temp->next != NULL) temp = temp->next;

    temp->next = newNode;
    return;
}
//utility function to find length of the list
int lengthOfLinkedList(node* head) {
    int length = 0;
    while(head != NULL) {
        ++length;
        head = head->next;
    }
    return length;
}
//utility function to reverse k nodes in the list
node* reverseKNodes(node* head,int k) {
    if(head == NULL||head->next == NULL) return head;

    int length = lengthOfLinkedList(head);

    node* dummyHead = new node(0);
    dummyHead->next = head;

    node* pre = dummyHead;
    node* cur;
    node* nex;

    while(length >= k) {
        cur = pre->next;
        nex = cur->next;
        for(int i=1;i<k;i++) {
            cur->next = nex->next;
            nex->next = pre->next;
            pre->next = nex;
            nex = cur->next;
        }
        pre = cur;
        length -= k;
    }
    return dummyHead->next;
}
//utility function to print the list
void printLinkedList(node* head) {
    while(head->next != NULL) {
        cout<<head->num<<"->";
        head = head->next;
    }
    cout<<head->num<<"\n";
}

int main() {
    node* head = NULL;
    int k = 3;
    insertNode(head,1);
    insertNode(head,2);
    insertNode(head,3);
    insertNode(head,4);
    insertNode(head,5);
    insertNode(head,6);
    insertNode(head,7);
    insertNode(head,8);

    cout<<"Original Linked List: "; printLinkedList(head);
    cout<<"After Reversal of k nodes: ";
    node* newHead = reverseKNodes(head,k);
    printLinkedList(newHead);

    return 0;
}
```

Output:

```
Original Linked List: 1->2->3->4->5->6->7->8
After Reversal of k nodes: 3->2->1->6->5->4->7->8
```

Time Complexity: O(N)

Reason: Nested iteration with O((N/k)*k) which makes it equal to O(N).

Space Complexity: O(1)

Reason: No extra data structures are used for computation.