Reverse a Linked List in groups of given size

Given a linked list, write a function to reverse every k nodes (where k is an input to the function).

```
Example:
Inputs:  1->2->3->4->5->6->7->8->NULL and k = 3
Output:  3->2->1->6->5->4->8->7->NULL.

Inputs:   1->2->3->4->5->6->7->8->NULL and k = 5
Output:  5->4->3->2->1->8->7->6->NULL.
```

We strongly recommend that you click here and code it yourself first, before moving on to the solution.

Algorithm: *reverse(head, k)*

1) Reverse the first sub-list of size k. While reversing keep track of the next node and previous node. Let the pointer to the next node be *next* and pointer to the previous node be *prev*. See this post for reversing a linked list.

2) *head->next = reverse(next, k) /** Recursively call for rest of the list and link the two sub-lists **/*

3) return *prev /* prev* becomes the new head of the list (see the diagrams of iterative method of this post) **/*

C/C++ Java Python

```c
// C program to reverse a linked list in groups of given size
#include<stdio.h>
#include<stdlib.h>

/* Link list node */
struct node
{
    int data;
    struct node* next;
};

/* Reverses the linked list in groups of size k and returns the
   pointer to the new head node. */
struct node *reverse (struct node *head, int k)
{
    struct node* current = head;
    struct node* next = NULL;
    struct node* prev = NULL;
    int count = 0;

    /*reverse first k nodes of the linked list */
    while (current != NULL && count < k)
    {
        next  = current->next;
        current->next = prev;
        prev = current;
        current = next;
        count++;
    }

    /* next is now a pointer to (k+1)th node
       Recursively call for the list starting from current.
       And make rest of the list as next of first node */
    if (next !=  NULL)
       head->next = reverse(next, k);

    /* prev is new head of the input list */
    return prev;
}

/* UTILITY FUNCTIONS */
/* Function to push a node */
void push(struct node** head_ref, int new_data)
{
    /* allocate node */
    struct node* new_node =
            (struct node*) malloc(sizeof(struct node));

    /* put in the data  */
    new_node->data  = new_data;

    /* link the old list off the new node */
    new_node->next = (*head_ref);

    /* move the head to point to the new node */
    (*head_ref)    = new_node;
}

/* Function to print linked list */
void printList(struct node *node)
{
    while (node != NULL)
    {
        printf("%d  ", node->data);
        node = node->next;
    }
}

/* Drier program to test above function*/
int main(void)
{
    /* Start with the empty list */
    struct node* head = NULL;

     /* Created Linked list is 1->2->3->4->5->6->7->8->9 */
     push(&head, 9);
     push(&head, 8);
     push(&head, 7);
     push(&head, 6);
     push(&head, 5);
     push(&head, 4);
     push(&head, 3);
     push(&head, 2);
     push(&head, 1);

     printf("\nGiven linked list \n");
     printList(head);
     head = reverse(head, 3);

     printf("\nReversed Linked list \n");
     printList(head);

     return(0);
}
```

Run on IDE

Output:

```
Given Linked List
1 2 3 4 5 6 7 8 9
Reversed list
3 2 1 6 5 4 9 8 7
```

Time Complexity: O(n) where n is the number of nodes in the given list.