Swap Kth node from beginning with Kth node from end in a Linked List

Given a singly linked list, swap kth node from beginning with kth node from end. **Swapping of data is not allowed, only pointers should be changed.** This requirement may be logical in many situations where the linked list data part is huge (For example student details line Name, RollNo, Address, ..etc). The pointers are always fixed (4 bytes for most of the compilers).

The problem seems simple at first look, but it has many interesting cases.

Let X be the kth node from beginning and Y be the kth node from end. Following are the interesting cases that must be handled.
1) Y is next to X
2) X is next to Y
3) X and Y are same
4) X and Y don't exist (k is more than number of nodes in linked list)

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

C++

```
// A C++ program to swap Kth node from beginning with kth node from end
#include <iostream>
#include <stdlib.h>
using namespace std;

// A Linked List node
struct node
{
    int data;
    struct node *next;
};

/* Utility function to insert a node at the beginning */
void push(struct node **head_ref, int new_data)
{
    struct node *new_node = (struct node *) malloc(sizeof(struct node));
    new_node->data = new_data;
    new_node->next = (*head_ref);
    (*head_ref) = new_node;
}

/* Utility function for displaying linked list */
void printList(struct node *node)
{
    while (node != NULL)
    {
        cout << node->data << " ";
        node = node->next;
    }
    cout << endl;
}

/* Utility function for calculating length of linked list */
int countNodes(struct node *s)
{
    int count = 0;
    while (s != NULL)
    {
        count++;
        s = s->next;
    }
    return count;
}

/* Function for swapping kth nodes from both ends of linked list */
void swapKth(struct node **head_ref, int k)
{
    // Count nodes in linked list
    int n = countNodes(*head_ref);

    // Check if k is valid
    if (n < k)  return;

    // If x (kth node from start) and y(kth node from end) are same
    if (2*k - 1 == n) return;

    // Find the kth node from beginning of linked list. We also find
    // previous of kth node because we need to update next pointer of
    // the previous.
    node *x = *head_ref;
    node *x_prev = NULL;
    for (int i = 1; i < k; i++)
    {
        x_prev = x;
        x = x->next;
    }

    // Similarly, find the kth node from end and its previous. kth node
    // from end is (n-k+1)th node from beginning
    node *y = *head_ref;
    node *y_prev = NULL;
    for (int i = 1; i < n-k+1; i++)
    {
        y_prev = y;
        y = y->next;
    }

    // If x_prev exists, then new next of it will be y. Consider the case
    // when y->next is x, in this case, x_prev and y are same. So the statement
    // "x_prev->next = y" creates a self loop. This self loop will be broken
    // when we change y->next.
    if (x_prev)
        x_prev->next = y;

    // Same thing applies to y_prev
    if (y_prev)
        y_prev->next = x;

    // Swap next pointers of x and y. These statements also break self
```

```
    // loop if x->next is y or y->next is x
    node *temp = x->next;
    x->next = y->next;
    y->next = temp;

    // Change head pointers when k is 1 or n
    if (k == 1)
        *head_ref = y;
    if (k == n)
        *head_ref = x;
}

// Driver program to test above functions
int main()
{
    // Let us create the following linked list for testing
    // 1->2->3->4->5->6->7->8
    struct node *head = NULL;
    for (int i = 8; i >= 1; i--)
       push(&head, i);

    cout << "Original Linked List: ";
    printList(head);

    for (int k = 1; k < 9; k++)
    {
        swapKth(&head, k);
        cout << "\nModified List for k = " << k << endl;
        printList(head);
    }

    return 0;
}
```

Java

```
// A Java program to swap kth node from the beginning with
// kth node from the end

class Node
{
    int data;
    Node next;
    Node(int d) { data = d;  next = null; }
}

class LinkedList
{
    Node head;

    /* Utility function to insert a node at the beginning */
    void push(int new_data)
    {
        Node new_node = new Node(new_data);
        new_node.next = head;
        head = new_node;
    }

    /* Utility function for displaying linked list */
    void printList()
    {
        Node node = head;
        while (node != null)
        {
            System.out.print(node.data + " ");
            node = node.next;
        }
        System.out.println("");
    }

    /* Utility function for calculating length of linked list */
    int countNodes()
    {
        int count = 0;
        Node s = head;
        while (s != null)
        {
            count++;
            s = s.next;
        }
        return count;
    }

    /* Function for swapping kth nodes from both ends of
       linked list */
    void swapKth(int k)
    {
        // Count nodes in linked list
        int n = countNodes();

        // Check if k is valid
        if (n < k)
            return;

        // If x (kth node from start) and y(kth node from end)
        // are same
        if (2*k - 1 == n)
            return;

        // Find the kth node from beginning of linked list.
        // We also find previous of kth node because we need
        // to update next pointer of the previous.
        Node x = head;
        Node x_prev = null;
        for (int i = 1; i < k; i++)
        {
            x_prev = x;
            x = x.next;
        }

        // Similarly, find the kth node from end and its
        // previous. kth node from end is (n-k+1)th node
        // from beginning
        Node y = head;
```

```
        Node y_prev = null;
        for (int i = 1; i < n - k + 1; i++)
        {
            y_prev = y;
            y = y.next;
        }

        // If x_prev exists, then new next of it will be y.
        // Consider the case when y->next is x, in this case,
        // x_prev and y are same. So the statement
        // "x_prev->next = y" creates a self loop. This self
        // loop will be broken when we change y->next.
        if (x_prev != null)
            x_prev.next = y;

        // Same thing applies to y_prev
        if (y_prev != null)
            y_prev.next = x;

        // Swap next pointers of x and y. These statements
        // also break self loop if x->next is y or y->next
        // is x
        Node temp = x.next;
        x.next = y.next;
        y.next = temp;

        // Change head pointers when k is 1 or n
        if (k == 1)
            head = y;

        if (k == n)
            head = x;
    }

    // Driver code to test above
    public static void main(String[] args)
    {
        LinkedList llist = new LinkedList();
        for (int i = 8; i >= 1; i--)
            llist.push(i);

        System.out.print("Original linked list: ");
        llist.printList();
        System.out.println("");

        for (int i = 1; i < 9; i++)
        {
            llist.swapKth(i);
            System.out.println("Modified List for k = " + i);
            llist.printList();
            System.out.println("");
        }
    }
}
```

Output:

```
Original Linked List: 1 2 3 4 5 6 7 8

Modified List for k = 1
8 2 3 4 5 6 7 1

Modified List for k = 2
8 7 3 4 5 6 2 1

Modified List for k = 3
8 7 6 4 5 3 2 1

Modified List for k = 4
8 7 6 5 4 3 2 1

Modified List for k = 5
8 7 6 4 5 3 2 1

Modified List for k = 6
8 7 3 4 5 6 2 1

Modified List for k = 7
8 2 3 4 5 6 7 1

Modified List for k = 8
1 2 3 4 5 6 7 8
```

Please note that the above code runs three separate loops to count nodes, find x and x prev, and to find y and y_prev. These three things can be done in a single loop. The code uses three loops to keep things simple and readable.