# Reverse a linked list

Given pointer to the head node of a linked list, the task is to reverse the linked list. We need to reverse the list by changing the links between nodes.

**Examples:**

Input: Head of following linked list
1->2->3->4->NULL
Output: Linked list should be changed to,
4->3->2->1->NULL

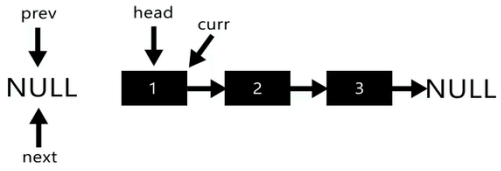
Input: Head of following linked list
1->2->3->4->5->NULL
Output: Linked list should be changed to,
5->4->3->2->1->NULL

Input: NULL
Output: NULL

Input: 1->NULL
Output: 1->NULL

## Iterative Method

1. Initialize three pointers prev as NULL, curr as head and next as NULL.
2. Iterate through the linked list. In loop, do following.
   // Before changing next of current,
   // store next node
   next = curr->next
   // Now change next of current
   // This is where actual reversing happens
   curr->next = prev
   // Move prev and curr one step forward
   prev = curr
   curr = next




```
while (current != NULL)
    {
        next  = current->next;
        current->next = prev;
        prev = current;
        current = next;
    }
    *head_ref = prev;
```

**Below is the implementation of the above approach:**

```
// Java program for reversing the linked list

class LinkedList {

	static Node head;

	static class Node {

		int data;
		Node next;

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

	/* Function to reverse the linked list */
	Node reverse(Node node)
	{
		Node prev = null;
		Node current = node;
		Node next = null;
		while (current != null) {
			next = current.next;
			current.next = prev;
			prev = current;
			current = next;
		}
		node = prev;
		return node;
	}

	// prints content of double linked list
	void printList(Node node)
	{
		while (node != null) {
			System.out.print(node.data + " ");
			node = node.next;
		}
	}
```

```
        // Driver Code
        public static void main(String[] args)
        {
                LinkedList list = new LinkedList();
                list.head = new Node(85);
                list.head.next = new Node(15);
                list.head.next.next = new Node(4);
                list.head.next.next.next = new Node(20);

                System.out.println("Given Linked list");
                list.printList(head);
                head = list.reverse(head);
                System.out.println("");
                System.out.println("Reversed linked list ");
                list.printList(head);
        }
}
```

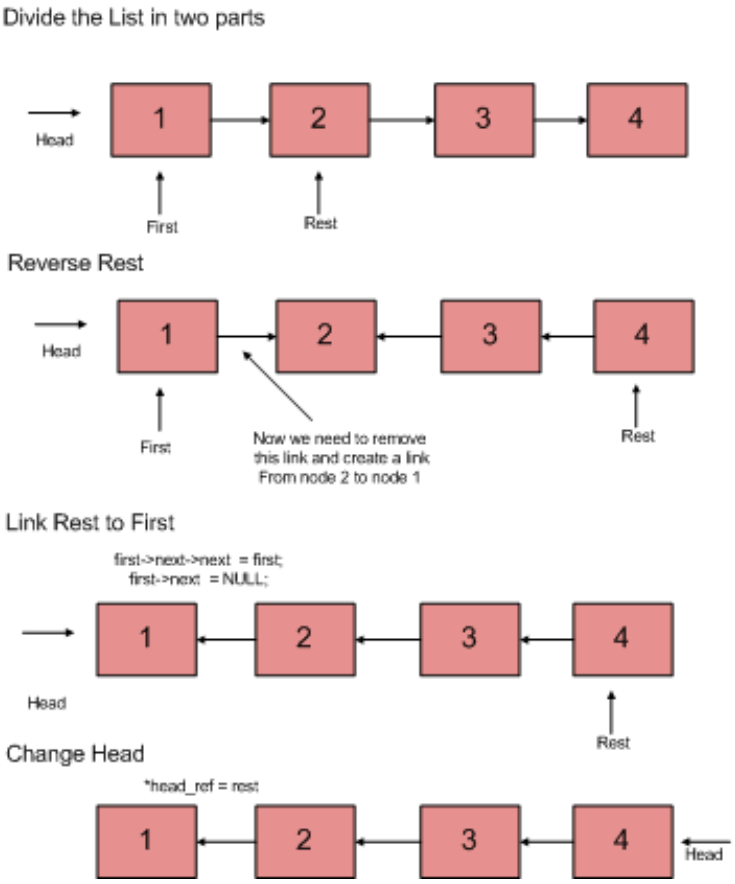
**Output:**

Given linked list
85 15 4 20
Reversed Linked list
20 4 15 85

Time Complexity: O(n)
Auxiliary Space: O(1)

## Recursive Method:

1) Divide the list in two parts - first node and
   rest of the linked list.
2) Call reverse for the rest of the linked list.
3) Link rest to first.
4) Fix head pointer




**Below is the implementation of the above approach:**

```
// Recursive Java program to reverse
// a linked list
class recursion {
	static Node head; /. / head of list

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

	static Node reverse(Node head)
	{
		if (head == null || head.next == null)
			return head;
```

```
/* reverse the rest list and put the first element at the end */
                Node rest = reverse(head.next);
                head.next.next = head;

                /* tricky step -- see the diagram */
                head.next = null;

                /* fix the head pointer */
                return rest;
        }
        /* Function to print linked list */
        static void print(){
                Node temp = head;
                while (temp != null) {
                        System.out.print(temp.data + " ");
                        temp = temp.next;
                }
                System.out.println();
        }

        static void push(int data)
        {
                Node temp = new Node(data);
                temp.next = head;
                head = temp;
        }

/* Driver program to test above function*/
public static void main(String args[])
{
        /* Start with the empty list */
        push(20);
        push(4);
        push(15);
        push(85);
        System.out.println("Given linked list");
        print();

        head = reverse(head);

        System.out.println("Reversed Linked list");
        print();
}
}
```

**Output:**

Given linked list
85 15 4 20
Reversed Linked list
20 4 15 85

Time Complexity: O(n)
Auxiliary Space: O(n)

## A Simpler and Tail Recursive Method

**Below is the implementation of this method:**

```
// Java program for reversing the Linked list

class LinkedList {

	static Node head;
	static class Node {
		int data;
		Node next;
		Node(int d){
			data = d;
			next = null;
		}
	}
	// A simple and tail recursive function to reverse
	// a linked list. prev is passed as NULL initially.
	Node reverseUtil(Node curr, Node prev)
	{
		/*If head is initially null OR list is empty*/
		if (head == null)
			return head;
		/* If last node mark it head*/
		if (curr.next == null) {
			head = curr;
			/* Update next to prev node */
			curr.next = prev;
			return head;
		}
```

```
            /* Save curr->next node for recursive call */
            Node next1 = curr.next;
            /* and update next ..*/
            curr.next = prev;
            reverseUtil(next1, curr);
            return head;
        }

        // prints content of double linked list
        void printList(Node node)
        {
            while (node != null) {
                System.out.print(node.data + " ");
                node = node.next;
            }
        }

        // Driver Code
        public static void main(String[] args)
        {
            LinkedList list = new LinkedList();
            list.head = new Node(1);
            list.head.next = new Node(2);
            list.head.next.next = new Node(3);
            list.head.next.next.next = new Node(4);
            list.head.next.next.next.next = new Node(5);
            list.head.next.next.next.next.next = new Node(6);
            list.head.next.next.next.next.next.next = new Node(7);
            list.head.next.next.next.next.next.next.next = new Node(8);

            System.out.println("Original Linked list ");
            list.printList(head);
            Node res = list.reverseUtil(head, null);
            System.out.println("");
            System.out.println("");
            System.out.println("Reversed linked list ");
            list.printList(res);
        }
}
```

**Output:**

Given linked list
1 2 3 4 5 6 7 8

Reversed linked list
8 7 6 5 4 3 2 1

Time Complexity: O(N)
As we do certain operation for every node of the linked list.

Auxiliary Space: O(1)
As constant extra space is used.

# Head Recursive Method:

## Using Stack:

**Algorithm : –**

1. Store the nodes(values and address) in the stack until all the values are entered.
2. Once all entries are done, Update the Head pointer to the last location(i.e the last value).
3. Start popping the nodes(value and address) and store them in the same order until the stack is empty.
4. Update the next pointer of last Node in the stack by NULL.

**Below is the implementation of the above approach:**

```
// Java program for above approach

import java.util.*;
class LinkedList {
        // Create a class Node to enter values and address in the list
        static class Node {
                int data;
                Node next;
        };
        static Node head = null;
        // Function to reverse the linked list
        static void reverseLL(){
```

```
        // Create a stack "s" of Node type
        Stack<Node> s = new Stack<>();
        Node temp = head;
        while (temp.next != null) {
                // Push all the nodes in to stack
                s.add(temp);
                temp = temp.next;
        }
        head = temp;
        while (!s.isEmpty()) {
                // Store the top value of stack in list
                temp.next = s.peek();
                // Pop the value from stack
                s.pop();
                // update the next pointer in the list
                temp = temp.next;
        }
        temp.next = null;
}

// Function to Display the elements in List
static void printlist(Node temp){
        while (temp != null) {
                System.out.print(temp.data + " ");
                temp = temp.next;
        }
}

// Program to insert back of the linked list
static void insert_back(int value){
        // we have used insertion at back method to enter
        // values in the list.(eg: head.1.2.3.4.Null)
        Node temp = new Node();
        temp.data = value;
        temp.next = null;
        // If *head equals to null
        if (head == null) {
                head = temp;
                return;
        } else {
                Node last_node = head;
                while (last_node.next != null)
                        last_node = last_node.next;
                last_node.next = temp;
                return;
```

```
            }
        }

        // Driver Code
        public static void main(String[] args)
        {
            insert_back(1);
            insert_back(2);
            insert_back(3);
            insert_back(4);
            System.out.print("Given linked list\n");
            printlist(head);
            reverseLL();
            System.out.print("\nReversed linked list\n");
            printlist(head);
        }
}
```

**Output:**

Given linked list
1 2 3 4
Reversed linked list
4 3 2 1

Time Complexity: O(N)
As we do certain operation for every node of the linked list.

Auxiliary Space: O(N)
Space is used to store the nodes in the stack.

## Using array:

**Algorithm :-**

1. Create a linked list.
2. Count the number of nodes present in the Linked List
3. Initialize an array with the size of the count.
4. Store the elements of the Linked list in array
5. Print the array from the last index to the first.

**Below is the implementation of the above approach:**

```
// Java program of the above approach

class LinkedList{
	static class node {
		int val;
		node next;
	};

	static node head = null;
	// code to count the no. of nodes
	static int count(node head)
	{
		node p = head;
		int k = 1;
		while (p != null) {
			p = p.next;
			k++;
		}
		return k;
	}

	// to reverse the linked list
	static node LL_reverse(node head) {
		node p = head;
		int i = count(head), j = 1;
		int[] arr = new int[i];
		while (i != 0 && p != null) {
			arr[j++] = p.val;
			p = p.next;
			i--;
		}
		j--;
		while (j != 0) // loop will break as soon as j=0
			System.out.print(arr[j--] + " ");
		return head;
	}
	// code to insert at end of LL
	static node insert_end(node head, int data)
	{
		node q = head;
		node p = new node();
		p.val = data;
```

```
            p.next = null;
            while (q.next != null)
                q = q.next;
            q.next = p;
            p.next = null;
            return head;
        }

        // create LL
        static node create_ll(node head, int data)
        {
            node p = new node();
            p.next = null;
            p.val = data;
            if (head == null) {
                head = p;
                p.next = null;
                return head;
            }
            else {
                head = insert_end(head, data);
                return head;
            }
        }

        public static void main(String[] args)
        {
            int i = 5, j = 1;
            while (i != 0) {
                head = create_LL(head, j++);
                i--;
            }
            head = LL_reverse(head);
        }
}
```

**Input :** 1->2->3->4->5
**Output:** 5->4->3->2->1

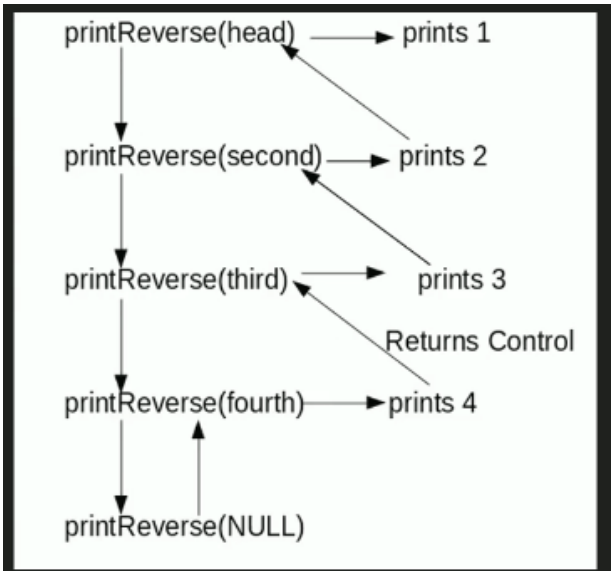
Time complexity: O(N) as we visit every node once.

Auxiliary Space: O(N) as extra space is used to store all the nodes in the array.

# Print reverse of a Linked List without actually reversing

Given a linked list, print reverse of it using a recursive function. For example, if the given linked list is 1->2->3->4, then output should be 4->3->2->1.
Note that the question is only about printing the reverse. To reverse the list itself see this

**Difficulty Level:** Rookie




**Algorithm :**

printReverse(head)
1. call print reverse for head->next
2. print head->data

**Implementation:**

```
// Java program to print reverse of a linked list
class LinkedList
{
        Node head; // head of list

        /* Linked list Node*/
```

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

/* Function to print reverse of linked list */
void printReverse(Node head)
{
        if (head == null) return;

        // print list of head node
        printReverse(head.next);

        // After everything else is printed
        System.out.print(head.data+" ");
}

/* Utility Functions */

/* Inserts a new Node at front of the list. */
public void push(int new_data)
{
        /* 1 & 2: Allocate the Node &
                        Put in the data*/
        Node new_node = new Node(new_data);

        /* 3. Make next of new Node as head */
        new_node.next = head;

        /* 4. Move the head to point to new Node */
        head = new_node;
}

/*Driver function to test the above methods*/
public static void main(String args[])
{
        // Let us create linked list 1->2->3->4
        LinkedList llist = new LinkedList();
        llist.push(4);
        llist.push(3);
        llist.push(2);
        llist.push(1);
        llist.printReverse(llist.head);
}
```

}

**Output:**
4 3 2 1

Time Complexity: O(n)
Auxiliary Space: O(n) for stack space since using recursion

## *Another approach:*

We can also perform the same action using a stack using iterative method.

**Algorithm :**

1. Store the values of the linked list in a stack.
2. Keep removing the elements from the stack and print them.

**Implementation:**

```
// Java program to print reverse of a linked list using iterative method
import java.util.*;
class LinkedList
{
	Node head; // head of list

	/* Linked list Node*/
	class Node {
		int data;
		Node next;
		Node(int d) {data = d; next = null; }
	}

	/* Function to print reverse of linked list */
	void printReverse(Node head)
	{
		Stack<Integer> st = new Stack<Integer>();
		Node curr = head;
		while(curr!=null)
		{
		st.push(curr.data);
		curr = curr.next;
		}
		while(st.isEmpty()==false)
```

```
            {
            System.out.print(st.peek() + " -> ");
            st.pop();
            }
    }

    /* Utility Functions */
    /* function to print the elements of the linked list*/
    void printList(Node head)
    {
            Node curr = head;
            while(curr!=null)
            {
                    System.out.print(curr.data + " -> ");
                    curr = curr.next;
            }
            System.out.println();
    }
    /* Inserts a new Node at front of the list. */
    public void push(int new_data)
    {
            /* 1 & 2: Allocate the Node &
                            Put in the data*/
            Node new_node = new Node(new_data);

            /* 3. Make next of new Node as head */
            new_node.next = head;

            /* 4. Move the head to point to new Node */
            head = new_node;
    }

    /*Driver function to test the above methods*/
    public static void main(String args[])
    {
            // Let us create linked list 1->2->3->4
            LinkedList llist = new LinkedList();
            llist.push(4);
            llist.push(3);
            llist.push(2);
            llist.push(1);
            llist.printList(llist.head);
            llist.printReverse(llist.head);
    }
}
```

**Output:**

1 -> 2 -> 3 -> 4 ->
4 -> 3 -> 2 -> 1 ->

Time Complexity: O(N)
As we are traversing the linked list only once.

Auxiliary Space: O(N)
The extra space is used in storing the elements in the stack.

# Iteratively Reverse a linked list using only 2 pointers (An Interesting Method)

Given pointer to the head node of a linked list, the task is to reverse the linked list.

**Examples:**

Input : Head of following linked list
1->2->3->4->NULL
Output : Linked list should be changed to,
4->3->2->1->NULL

Input : Head of following linked list
1->2->3->4->5->NULL
Output : Linked list should be changed to,
5->4->3->2->1->NULL

We have seen how to reverse a linked list in article Reverse a linked list. In iterative method we had used 3 pointers prev, cur and next. Below is an interesting approach that uses only two pointers. The idea is to use XOR to swap pointers.

**Below is the implementation of the above approach:**

```
// Java program to reverse a linked
// list using two pointers.
import java.util.*;

class Main {

	// Link list node
	static class Node {
		int data;
		Node next;
	};

	static Node head_ref = null;

	// Function to reverse the linked
	// list using 2 pointers
	static void reverse(){
		Node prev = null;
		Node current = head_ref;

		// At last prev points to new head
		while (current != null) {
			Node next = current.next;
			current.next = prev;
			prev = current;
			current = next;
		}
		head_ref = prev;
	}
	// Function to push a node
	static void push(int new_data){

		// Allocate node
		Node new_node = new 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
        static void printList(){
                Node temp = head_ref;
                while (temp != null) {
                        System.out.print(temp.data + " ");
                        temp = temp.next;
                }
        }

        // Driver code
        public static void main(String[] args){
                push(20);
                push(4);
                push(15);
                push(85);

                System.out.print("Given linked list\n");
                printList();
                reverse();
                System.out.print("\nReversed Linked list \n");
                printList();
        }
}
```

**Output:**
Given linked list
85  15  4  20

Reversed Linked list
20  4  15  85
Time Complexity: O(n)
Auxiliary Space: O(1) since using space for prev and next

## Alternate Solution :

```
// Java program to reverse a linked
// list using two pointers.
import java.util.*;

class Main {

	// Link list node
	static class Node {
		int data;
		Node next;
	};

	static Node head_ref = null;

	// Function to reverse the linked
	// list using 2 pointers
	static void reverse()
	{
		Node next;
		Node curr = head_ref;
		while (curr.next != null) {
			next = curr.next;
			curr.next = next.next;
			next.next = head_ref;
			head_ref = next;
		}
	}

	// Function to push a node
	static void push(int new_data)
	{

		// Allocate node
		Node new_node = new 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
        static void printList()
        {
            Node temp = head_ref;
            while (temp != null) {
                System.out.print(temp.data + " ");
                temp = temp.next;
            }
        }

        // Driver code
        public static void main(String[] args)
        {
            push(20);
            push(4);
            push(15);
            push(85);

            System.out.print("Given linked list\n");
            printList();
            reverse();
            System.out.print("\nReversed Linked list \n");
            printList();
        }
}
```

**Output:**

Given linked list
85 15 4 20
Reversed Linked list
20 4 15 85

Time Complexity : O(n)
Auxiliary Space: O(1)

# Reverse a Linked List in groups of given size | Set 1

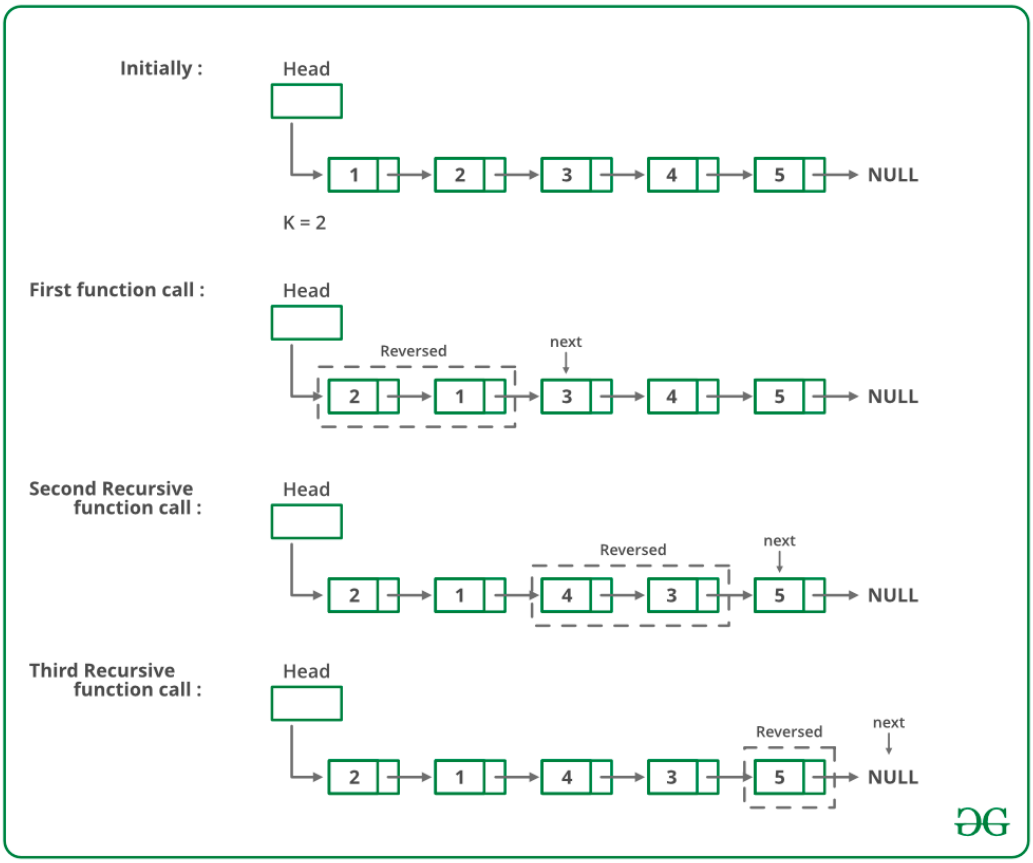
Given a linked list, write a function to reverse every k nodes (where k is an input to the function).

**Example:**

Input: 1->2->3->4->5->6->7->8->NULL, K = 3
Output: 3->2->1->6->5->4->8->7->NULL
Input: 1->2->3->4->5->6->7->8->NULL, K = 5
Output: 5->4->3->2->1->8->7->6->NULL

**Algorithm:** reverse(head, k)

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.
head->next = reverse(next, k) ( Recursively call for rest of the list and link the two sub-lists )
Return prev ( prev becomes the new head of the list (see the diagrams of an iterative method of this post )
Below is image shows how the reverse function works:



**Below is the implementation of the above approach:**

```
// Java program to reverse a linked list in groups of
// given size
class LinkedList {
	Node head; // head of list

	/* Linked list Node*/
	class Node {
		int data;
		Node next;
		Node(int d)
		{
			data = d;
			next = null;
		}
	}

	Node reverse(Node head, int k)
	{
		if(head == null)
		return null;
		Node current = head;
		Node next = null;
		Node prev = null;

		int count = 0;

		/* Reverse first k nodes of linked list */
		while (count < k && current != null) {
			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 now head of input list
        return prev;
}

/* Utility functions */

/* Inserts a new Node at front of the list. */
public void push(int new_data)
{
        /* 1 & 2: Allocate the Node &
                        Put in the data*/
        Node new_node = new Node(new_data);

        /* 3. Make next of new Node as head */
        new_node.next = head;

        /* 4. Move the head to point to new Node */
        head = new_node;
}

/* Function to print linked list */
void printList(){
        Node temp = head;
        while (temp != null) {
                System.out.print(temp.data + " ");
                temp = temp.next;
        }
        System.out.println();
}

/* Driver program to test above functions */
public static void main(String args[]){
        LinkedList llist = new LinkedList();

        /* Constructed Linked List is 1->2->3->4->5->6->
        7->8->8->9->null */
        llist.push(9);  //(llist= linked list)
        llist.push(8);
        llist.push(7);
        llist.push(6);
        llist.push(5);
        llist.push(4);
        llist.push(3);
        llist.push(2);
        llist.push(1);
```

```
                System.out.println("Given Linked List");
                llist.printList();

                llist.head = llist.reverse(llist.head, 3);

                System.out.println("Reversed list");
                llist.printList();
        }
}
```

**Output :**

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

**Complexity Analysis:**

Time Complexity: O(n).
Traversal of list is done only once and it has ‘n’ elements.
Auxiliary Space: O(n/k).

For each Linked List of size n, n/k or (n/k)+1 calls will be made during the recursion.

We can solve this question in O(1) Space Complexity.

**Approach – 2 Space Optimized – Iterative**

The following steps are required for this Algorithm:

1. Create a dummy node and point it to the head of input i.e dummy->next = head.
2. Calculate the length of the linked list which takes O(N) time, where N is the length of the linked list.
3. Initialize three-pointers prev, curr, next to reverse k elements for every group.
4. Iterate over the linked lists till next!=NULL.
5. Points curr to the prev->next and next to the curr next.
6. Then, Using the inner for loop reverse the particular group using these four steps:

curr->next = next->next
next->next = prev->next
prev->next = next
next = curr->next

7. This for loop runs for k-1 times for all groups except the last remaining element, for the last remaining element it runs for the remaining length of the linked list – 1.
8. Decrement count after for loop by k count -= k, to determine the length of the remaining linked list.
9. Change prev position to curr, prev = curr.

**Here is the code for the above algorithm.**

```
import java.util.*;

// Linked List Node
class Node {
	int data;
	Node next;
	Node(int a)
	{
		data = a;
		next = null;
	}
}

class GFG {
	// utility function to insert node in the list
	static Node push(Node head, int val) {
		Node newNode = new Node(val);
		if (head == null) {
			head = newNode;
			return head;
		}

		Node temp = head;
		while (temp.next != null)
			temp = temp.next;

		temp.next = newNode;
		return head;
	}

	// utility function to reverse k nodes in the list
	static Node reverse(Node head, int k)
	{
		// If head is NULL or K is 1 then return head
		if (head == null || head.next == null)
			return head;
```

```
        // creating dummy node
        Node dummy = new Node(-1);
        dummy.next = head;

        // Initializing three points prev, curr, next
        Node prev = dummy;
        Node curr = dummy;
        Node next = dummy;

        // Calculating the length of linked list
        int count = 0;
        while (curr != null) {
            count++;
            curr = curr.next;
        }

        // Iterating till next is not NULL
        while (next != null) {
            curr = prev.next; // Curr position after every
                              // reverse group
            next = curr.next; // Next will always next to
                              // curr
            int toLoop
                = count > k
                    ? k
                    : count - 1; // toLoop will set to
                                 // count - 1 in case of
                                 // remaining element

            for (int i = 1; i < toLoop; i++) {
                // 4 steps as discussed above
                curr.next = next.next;
                next.next = prev.next;
                prev.next = next;
                next = curr.next;
            }
            prev = curr; // Setting prev to curr
            count -= k; // Update count
        }
        return dummy.next; // dummy -> next will be our new
                           // head for output linked
        // list
}
// utility function to print the list
```

```
	static void print(Node head)
	{
		while (head.next != null) {
			System.out.print(head.data + " ");
			head = head.next;
		}
		System.out.println(head.data);
	}

	public static void main(String args[])
	{
		Node head = null;
		int k = 3;
		head = push(head, 1);
		head = push(head, 2);
		head = push(head, 3);
		head = push(head, 4);
		head = push(head, 5);
		head = push(head, 6);
		head = push(head, 7);
		head = push(head, 8);
		head = push(head, 9);

		System.out.println("Given Linked List");
		print(head);
		System.out.println("Reversed list");
		Node newHead = reverse(head, k);
		print(newHead);
	}
}
```

**Output :**
Given linked list
1 2 3 4 5 6 7 8 9
Reversed Linked list
3 2 1 6 5 4 9 8 7

**Complexity Analysis :**
Time Complexity: O(N) : While loop takes O(N/K) time and inner for loop takes O(K) time.
So N/K * K = N. Therefore TC O(N)

Space Complexity: O(1) : No extra space is used.