Segregate even and odd nodes in a Linked List

Given a Linked List of integers, write a function to modify the linked list such that all even numbers appear before all the odd numbers in the modified linked list. Also, keep the order of even and odd numbers same.

Examples:
Input: 17->15->8->12->10->5->4->1->7->6->NULL
Output: 8->12->10->4->6->17->15->5->1->7->NULL

Input: 8->12->10->5->4->1->6->NULL
Output: 8->12->10->4->6->5->1->NULL

// If all numbers are even then do not change the list
Input: 8->12->10->NULL
Output: 8->12->10->NULL

// If all numbers are odd then do not change the list
Input: 1->3->5->7->NULL
Output: 1->3->5->7->NULL

Method 1
The idea is to get pointer to the last node of list. And then traverse the list starting from the head node and move the odd valued nodes from their current position to end of the list.

Thanks to blunderboy for suggesting this method.

Algorithm:
…1) Get pointer to the last node.
…2) Move all the odd nodes to the end.
……..a) Consider all odd nodes before the first even node and move them to end.
……..b) Change the head pointer to point to the first even node.
……..b) Consider all odd nodes after the first even node and move them to the end.

C/C++

```
// C/C++ program to segregate even and odd nodes in a
// Linked List
#include <stdio.h>
#include <stdlib.h>

/* a node of the singly linked list */
struct node
{
    int data;
    struct node *next;
};

void segregateEvenOdd(struct node **head_ref)
{
    struct node *end = *head_ref;
    struct node *prev = NULL;
    struct node *curr = *head_ref;

    /* Get pointer to the last node */
    while (end->next != NULL)
        end = end->next;

    struct node *new_end = end;

    /* Consider all odd nodes before the first even node
       and move then after end */
    while (curr->data %2 != 0 && curr != end)
    while (curr->data %2 != 0 && curr != end)
    {
        new_end->next = curr;
        curr = curr->next;
        new_end->next->next = NULL;
        new_end = new_end->next;
    }

    // 10->8->17->17->15
    /* Do following steps only if there is any even node */
    if (curr->data%2 == 0)
    {
        /* Change the head pointer to point to first even node */
        *head_ref = curr;

        /* now current points to the first even node */
        while (curr != end)
        {
            if ( (curr->data)%2 == 0 )
            {
                prev = curr;
                curr = curr->next;
            }
            else
            {
                /* break the link between prev and current */
                prev->next = curr->next;

                /* Make next of curr as NULL  */
                curr->next = NULL;

                /* Move curr to end */
                new_end->next = curr;

                /* make curr as new end of list */
                new_end = curr;

                /* Update current pointer to next of the moved node */
                curr = prev->next;
            }
        }
    }

    /* We must have prev set before executing lines following this
       statement */
    else prev = curr;

    /* If there are more than 1 odd nodes and end of original list is
      odd then move this node to end to maintain same order of odd
      numbers in modified list */
    if (new_end!=end && (end->data)%2 != 0)
    {
        prev->next = end->next;
        end->next = NULL;
        new_end->next = end;
    }
    return;
}

/* UTILITY FUNCTIONS */
/* Function to insert a node at the beginning  */
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 nodes in a given linked list */
void printList(struct node *node)
{
    while (node!=NULL)
    {
        printf("%d ", node->data);
        node = node->next;
    }
}

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

    /* Let us create a sample linked list as following
      0->2->4->6->8->10->11 */

    push(&head, 11);
    push(&head, 10);
    push(&head, 8);
    push(&head, 6);
    push(&head, 4);
    push(&head, 2);
    push(&head, 0);

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

    segregateEvenOdd(&head);

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

    return 0;
}
```

Java

```
// Java program to segregate even and odd nodes in 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;
        }
    }

    void segregateEvenOdd()
    {
        Node end = head;
        Node prev = null;
        Node curr = head;

        /* Get pointer to last Node */
        while (end.next != null)
```

```
            end = end.next;

        Node new_end = end;

        // Consider all odd nodes before getting first eve node
        while (curr.data %2 !=0 && curr != end)
        {
            new_end.next = curr;
            curr = curr.next;
            new_end.next.next = null;
            new_end = new_end.next;
        }

        // do following steps only if there is an even node
        if (curr.data %2 ==0)
        {
            head = curr;

            // now curr points to first even node
            while (curr != end)
            {
                if (curr.data % 2 == 0)
                {
                    prev = curr;
                    curr = curr.next;
                }
                else
                {
                    /* Break the link between prev and curr*/
                    prev.next = curr.next;

                    /* Make next of curr as null */
                    curr.next = null;

                    /*Move curr to end */
                    new_end.next = curr;

                    /*Make curr as new end of list */
                    new_end = curr;

                    /*Update curr pointer */
                    curr = prev.next;
                }
            }
        }

        /* We have to set prev before executing rest of this code */
        else prev = curr;

        if (new_end != end && end.data %2 != 0)
        {
            prev.next = end.next;
            end.next = null;
            new_end.next = end;
        }
    }

    /*  Given a reference (pointer to pointer) to the head
        of a list and an int, push a new node on the front
        of the list. */
    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;
    }
    // Utility function to print a linked list
    void printList()
    {
        Node temp = head;
        while(temp != null)
        {
            System.out.print(temp.data+" ");
            temp = temp.next;
        }
        System.out.println();
    }


    /* Drier program to test above functions */
    public static void main(String args[])
    {
        LinkedList llist = new LinkedList();
        llist.push(11);
        llist.push(10);
        llist.push(8);
        llist.push(6);
        llist.push(4);
        llist.push(2);
        llist.push(0);
        System.out.println("Origional Linked List");
        llist.printList();

        llist.segregateEvenOdd();

        System.out.println("Modified Linked List");
        llist.printList();
    }
} /* This code is contributed by Rajat Mishra */
```

Output:

```
Original Linked list 0 2 4 6 8 10 11
Modified Linked list 0 2 4 6 8 10 11
```

Time complexity: O(n)

Method 2
The idea is to split the linked list into two: one containing all even nodes and other containing all odd nodes. And finally attach the odd node linked list after the even node linked list.
To split the Linked List, traverse the original Linked List and move all odd nodes to a separate Linked List of all odd nodes. At the end of loop, the original list will have all the even nodes and the odd node list will have all the odd nodes. To keep the ordering of all nodes same, we must insert all the odd nodes at the end of the odd node list. And to do that in constant time, we must keep track of last pointer in the odd node list.

Time complexity: O(n)