Given a linked list, reverse alternate nodes and append at the end

Given a linked list, reverse alternate nodes and append them to end of list. Extra allowed space is O(1)

Examples

```
Input List:  1->2->3->4->5->6
Output List: 1->3->5->6->4->2

Input List:  12->14->16->18->20
Output List: 12->16->20->18->14
```

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

The idea is to maintain two linked lists, one list of all odd positioned nodes (1, 3, 5 in above example) and other list of all even positioned nodes (6, 4 and 2 in above example). Following are detailed steps.

1) Traverse the given linked list which is considered as odd list. Do following for every visited node.

.....**a)** If the node is even node, remove it from odd list and add it to the front of even node list. Nodes are added at front to keep the reverse order.

2) Append the even node list at the end of odd node list.

C

```
#include<stdio.h>
#include<stdlib.h>

/* A linked list node */
struct node
{
    int data;
    struct node *next;
};

/* Function to reverse all even positioned node and append at the end
   odd is the head node of given linked list */
void rearrange(struct node *odd)
{
    // If linked list has less than 3 nodes, no change is required
    if (odd == NULL || odd->next == NULL || odd->next->next == NULL)
        return;

    // even points to the beginning of even list
    struct node *even = odd->next;

    // Remove the first even node
    odd->next = odd->next->next;

    // odd points to next node in odd list
    odd = odd->next;

    // Set terminator for even list
    even->next = NULL;

    // Traverse the  list
    while (odd && odd->next)
    {
       // Store the next node in odd list
```

```
        struct node *temp = odd->next->next;

        // Link the next even node at the beginning of even list
        odd->next->next = even;
        even = odd->next;

        // Remove the even node from middle
        odd->next = temp;

        // Move odd to the next odd node
        if (temp != NULL)
          odd = temp;
    }

    // Append the even list at the end of odd list
    odd->next = even;
}

/* Function to add a node at the beginning of Linked List */
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;
}

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

/* Druver program to test above function */
int main()
{
    struct node *start = NULL;

    /* The constructed linked list is:
     1->2->3->4->5->6->7 */
    push(&start, 7);
    push(&start, 6);
    push(&start, 5);
    push(&start, 4);
    push(&start, 3);
    push(&start, 2);
    push(&start, 1);

    printf("\n Linked list before calling  rearrange() ");
    printList(start);

    rearrange(start);

    printf("\n Linked list after calling  rearrange() ");
    printList(start);

    return 0;
}
```

Java

```
// Java program to reverse alternate nodes of a linked list
// and append at the end

class LinkedList {

    static Node head;
```

```
static Node head;

static class Node {

    int data;
    Node next;

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

/* Function to reverse all even positioned node and append at the end
 odd is the head node of given linked list */
void rearrange(Node odd) {

    // If linked list has less than 3 nodes, no change is required
    if (odd == null || odd.next == null || odd.next.next == null) {
        return;
    }

    // even points to the beginning of even list
    Node even = odd.next;

    // Remove the first even node
    odd.next = odd.next.next;

    // odd points to next node in odd list
    odd = odd.next;

    // Set terminator for even list
    even.next = null;

    // Traverse the  list
    while (odd != null && odd.next != null) {

        // Store the next node in odd list
        Node temp = odd.next.next;

        // Link the next even node at the beginning of even list
        odd.next.next = even;
        even = odd.next;

        // Remove the even node from middle
        odd.next = temp;

        // Move odd to the next odd node
        if (temp != null) {
            odd = temp;
        }
    }

    // Append the even list at the end of odd list
    odd.next = even;
}

/* Function to print nodes in a given linked list */
void printList(Node node) {
    while (node != null) {
        System.out.print(node.data + " ");
        node = node.next;
    }
}

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);

        System.out.println("Linked list before calling rearrange : ");
        list.printList(head);

        System.out.println("");
        list.rearrange(head);

        System.out.println("Linked list after calling rearrange : ");
        list.printList(head);

    }
}
```

Python

```
# Python program to reverse alternate nodes and append
# at end
# Extra space allowed - O(1)

# Node Class
class Node:

    # Constructor to initialize the node object
    def  __init__(self, data):
        self.data = data
        self.next = None

# Linked list class contains node object
class LinkedList:

    # Constructor to initialize head
    def __init__(self):
        self.head = None

    # Function to insert a new node at the beginning
    def push(self, new_data):
        new_node = Node(new_data)
        new_node.next = self.head
        self.head = new_node


    def printList(self):
        temp = self.head
        while(temp):
            print temp.data,
            temp = temp.next

    def rearrange(self):

        # If linked list has less than 3 nodes, no change
        # is required
        odd = self.head
        if (odd is None or odd.next is None or
            odd.next.next is None):
            return

        # Even points to the beginning of even list
        even = odd.next

        # Remove the first even node
        odd.next = odd.next.next

        # Odd points to next node in odd list
        odd = odd.next

        # Set terminator for even list
        even.next = None
```

```
        # Traverse the list
        while (odd and odd.next):
            # Store the next node in odd list
            temp = odd.next.next

            # Link the next even node at the beginning
            # of even list
            odd.next.next = even
            even = odd.next

            # Remove the even node from middle
            odd.next = temp

            # Move odd to the next odd node
            if temp is not None:
                odd = temp

        # Append the even list at the end of odd list
        odd.next = even

# Code execution starts here
if __name__ == '__main__':
    start = LinkedList()

    #The constructed linked list is ;
    # 1->2->3->4->5->6->7
    start.push(7)
    start.push(6)
    start.push(5)
    start.push(4)
    start.push(3)
    start.push(2)
    start.push(1)

    print "Linked list before calling  rearrange() "
    start.printList()

    start.rearrange()

    print "\nLinked list after calling  rearrange()"
    start.printList()

# This code is contributed by NIkhil Kumar Singh(nickzuck_007)
```

Output:

```
 Linked list before calling  rearrange() 1 2 3 4 5 6 7
 Linked list after calling  rearrange()  1 3 5 7 6 4 2
```

Time Complexity: The above code simply traverses the given linked list. So time complexity is O(n)

Auxiliary Space: O(1)