Segregate even and odd nodes in a Linked List

Given a **Linked List** of integers, The task is to modify the linked list such that all **even numbers** appear before all the **odd numbers** in the modified linked list. Also, **preserve** the order of even and odd numbers.

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*
Explanation*: In the output list, we have all the even nodes first (in the same order as input list, then all the odd nodes of the list (in the same order as input list).*

Input: *8->12->10->5->4->1->6->NULL*
Output: *8->12->10->4->6->5->1->NULL*
Explanation*: We do not change the list as all the numbers are even.*

[Naive Approach] By Creating Two New Linked List – O(n) Time and O(n) Space:

*The idea is to initialize pointers to track **even** and **odd** list separately. Traverse the list list end. While traversing the **odd data** node should be appended in **odd list pointer and even data** node should be appended in **even list pointer** and update the original list's head to the **even list's** head.*

Step-by-step implementation:

- Initialize pointers for even and odd list ,
  say **evenStart** , **evenEnd** and **oddStart** , **oddEnd**.
- Separate nodes into **even and odd** lists during traversal.
- Link the end of the **even** list to the start of the **odd** list.
- Update the **original list's head** to the **even** list's head.

```
#include <bits/stdc++.h>
using namespace std;
```

```
class Node {
  public:
    int data;
    Node *next;
    Node(int new_data) {
        data = new_data;
        next = nullptr;
    }
};

// Function to segregate even and odd nodes and return
// the head of the new list.
Node *segregateEvenOdd(Node *head) {

    // Starting node of list having even values.
    Node *evenStart = nullptr;
    Node *evenEnd = nullptr;

    // Same for the odd list.
    Node *oddStart = nullptr;
    Node *oddEnd = nullptr;

    // Node to traverse the list.
    Node *currNode = head;

    while (currNode != nullptr) {
        int val = currNode->data;

        // If current value is even, add it to the
        // even values list.
        if (val % 2 == 0) {
            if (evenStart == nullptr) {
                evenStart = currNode;
                evenEnd = evenStart;
            }
            else {
                evenEnd->next = currNode;
                evenEnd = evenEnd->next;
            }
        }
        else {

            // If current value is odd, add it to
              // the odd values list.
            if (oddStart == nullptr) {
                oddStart = currNode;
                oddEnd = oddStart;
            }
            else {
```

```
                oddEnd->next = currNode;
                oddEnd = oddEnd->next;
            }
        }

        // Move to the next node.
        currNode = currNode->next;
    }

    // If either odd list or even list is empty,
      // return the head as is.
    if (oddStart == nullptr || evenStart == nullptr)
        return evenStart;

    // Add odd list after even list.
    evenEnd->next = oddStart;
    oddEnd->next = nullptr;

    // Return the head of the modified list.
    return evenStart;
}

void printList(Node *node) {
    while (node != nullptr) {
        cout << node->data << " ";
        node = node->next;
    }
}

int main() {
    // 0->1->4->6->9->10->11
    Node *head = new Node(0);
    head->next = new Node(1);
    head->next->next = new Node(4);
    head->next->next->next = new Node(6);
    head->next->next->next->next = new Node(9);
    head->next->next->next->next->next = new Node(10);
    head->next->next->next->next->next->next = new Node(11);

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

    head = segregateEvenOdd(head);

    cout << "\nModified Linked list: ";
    printList(head);

    return 0;
}
```

[Expected Approach] By Adding Odd Nodes at End – O(n) Time and O(1) Space:

*The idea is traverse the list to find the **end node**, move **odd nodes** to the end while preserving the relative order of both **even and odd** nodes.*

Step-by-step implementation:

- Get a pointer to the **end** node.
- Move all the odd nodes to the end.
    - Consider all **odd nodes** before the first **even node** and move them to end.
    - Change the **head** pointer to point to the **first even** node.
    - Consider all **odd** nodes after the first **even** node and move them to the end.

```
#include <bits/stdc++.h>
using namespace std;
class Node {
  public:
    int data;
    Node *next;
    Node(int d) {
        data = d;
        next = nullptr;
    }
};

Node *segregateEvenOdd(Node *head) {
    Node *end = head;
    Node *prev = nullptr;
    Node *curr = head;

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

    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) {
        new_end->next = curr;
```

```
        curr = curr->next;
        new_end->next->next = nullptr;
        new_end = new_end->next;
    }

    // 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 = 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 = nullptr;

                // 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 = nullptr;
        new_end->next = end;
    }
    return head;
}

void printList(Node *node) {
    while (node != nullptr) {
        cout << node->data << " ";
        node = node->next;
    }
}

int main() {

    // Let us create a sample linked list as following
    // 0->2->4->6->9->10->11
    Node *head = new Node(0);
    head->next = new Node(1);
    head->next->next = new Node(4);
    head->next->next->next = new Node(6);
    head->next->next->next->next = new Node(9);
    head->next->next->next->next->next = new Node(10);
    head->next->next->next->next->next->next = new Node(11);

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

    head = segregateEvenOdd(head);

    cout << "\nModified Linked list: ";
    printList(head);

    return 0;
}
```