Delete last node of Singly linked list

Given a linked list, the task is to remove the last node of the linked list and update the head pointer of the linked list.

Examples:

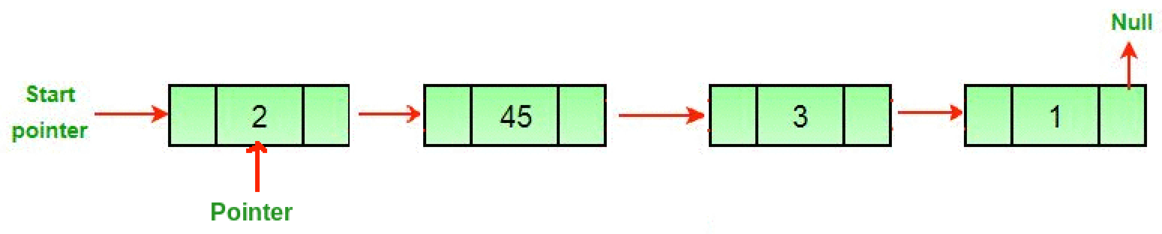
```
Input: 1 -> 2 -> 3 -> 4 -> 5 -> NULL
Output: 1 -> 2 -> 3 -> 4 -> NULL

Explanation: The last node of the linked list
is 5, so 5 is deleted.

Input: 2 -> 4 -> 6 -> 8 -> 33 -> 67 -> NULL
Output:2 -> 4 -> 6 -> 8 -> 33 -> NULL

Explanation: The last node of the linked list
is 67, so 67 is deleted.
```

Approach: To delete the last node of a linked list, find the second last node and make the next pointer of that node null.



Algorithm:

1. If the first node is *null* or there is only one node, then they return null.

```
if headNode == null then return null
if headNode.nextNode == null then free
head and return null
```

2. Create an extra space *secondLast*, and traverse the linked list till the second last node.

```
while secondLast.nextNode.nextNode != null
            secondLast =secondLast.nextNode
```

3. delete the last node, i.e. the next node of the second last node delete(*secondLast*.nextNode), and set the value of the next second-last node to null.

Implementation:

```
// CPP program to remove last node of
// linked list.
#include <iostream>
using namespace std;

/* Link list node */
struct Node {
    int data;
    struct Node* next;
};

/* Function to remove the last node
of the linked list */
Node* removeLastNode(struct Node* head)
{
    if (head == NULL)
        return NULL;

    if (head->next == NULL) {
        delete head;
        return NULL;
    }

    // Find the second last node
    Node* second_last = head;
    while (second_last->next->next != NULL)
        second_last = second_last->next;

    // Delete last node
    delete (second_last->next);

    // Change next of second last
    second_last->next = NULL;

    return head;
}

// Function to push node at head
void push(struct Node** head_ref, int new_data)
{
    struct Node* new_node = new Node;
    new_node->data = new_data;
    new_node->next = (*head_ref);
    (*head_ref) = new_node;
}

// Driver code
int main()
{
    /* Start with the empty list */
    Node* head = NULL;

    /* Use push() function to construct
    the below list 8 -> 23 -> 11 -> 29 -> 12 */
    push(&head, 12);
    push(&head, 29);
    push(&head, 11);
    push(&head, 23);
    push(&head, 8);

    head = removeLastNode(head);
    for (Node* temp = head; temp != NULL; temp = temp->next)
        cout << temp->data << " ";

    return 0;
}
```

Output:

```
8 23 11 29
```

Complexity Analysis:

- **Time Complexity:** O(n). The algorithm involves traversal of the linked list till its end, so the time complexity required is *O(n)*.
- **Space Complexity:** O(1). No extra space is required, so the space complexity is constant.