***Delete Last 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.

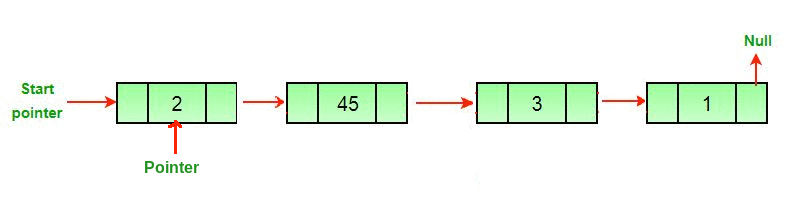
**Input:** 2 -> 4 -> 6 -> 8 -> 33 -> 67 -> NULL

**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:**

C++Java

// Java program to remove last node of

// linked list.

class GFG {

// Link list node /

static class Node {

int data;

Node next;

};

// Function to remove the last node

// of the linked list /

static Node removeLastNode(Node head)

{

if (head == null)

return null;

if (head.next == null) {

return null;

}

// Find the second last node

Node second\_last = head;

while (second\_last.next.next != null)

second\_last = second\_last.next;

// Change next of second last

second\_last.next = null;

return head;

}

// Function to push node at head

static Node push(Node head\_ref, int new\_data)

{

Node new\_node = new Node();

new\_node.data = new\_data;

new\_node.next = (head\_ref);

(head\_ref) = new\_node;

return head\_ref;

}

// Driver code

public static void main(String args[])

{

// Start with the empty list /

Node head = null;

// Use push() function to construct

// the below list 8 . 23 . 11 . 29 . 12 /

head = push(head, 12);

head = push(head, 29);

head = push(head, 11);

head = push(head, 23);

head = push(head, 8);

head = removeLastNode(head);

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

System.out.print(temp.data + " ");

}

}

**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.