Deletion in Singly Linked List

> *Given the head pointer pointing to the Head of a singly linked list and a node to be deleted from the list. Delete the given node from the Linked List.*

Like inserting a node in a linked list, there can be many situations when it comes to deleting a Node from a Linked List. Some of the most frequent such situations are:

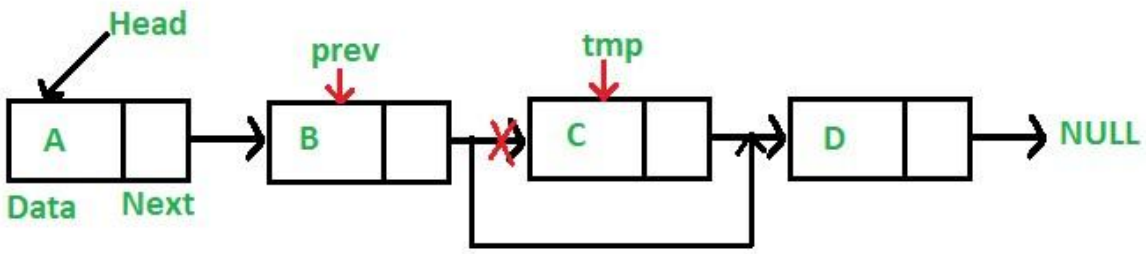
- Given the data value of a Node, delete the first occurrence of that data in the list.
- Given the position of a node, delete the node present at the given position in the list.
- Given a pointer to the node to be deleted, delete the node.

Let us look at each one of these situations and their solutions with complete explanations:

- **Deleting the first occurrence of a given Data Value:** Deleting a node by its value can be done by traversing the list till the node just before the node with the value as the given key. Once the node just before the node to be deleted is found. Update its next pointer to point to the next of its next node.

  That is:

  1. Find previous node of the node to be deleted.
  2. Change the next of previous node.
  3. Free memory for the node to be deleted.



Below is the implementation of this approach:

C++ Java

```
/* Given a reference (pointer to pointer) to the head of a list
   and a key, deletes the first occurrence of key in linked list */
void deleteNode(struct Node **head_ref, int key)
{
    // Store head node
    struct Node* temp = *head_ref, *prev;
    // If head node itself holds the key to be deleted
    if (temp != NULL && temp->data == key)
    {
        *head_ref = temp->next;   // Changed head
        free(temp);               // free old head
        return;
    } // Search for the key to be deleted, keep track of the
    // previous node as we need to change 'prev->next'
    while (temp != NULL && temp->data != key)
    {
        prev = temp;
        temp = temp->next;
    }
    // If key was not present in linked list
    if (temp == NULL) return;
    // Unlink the node from linked list
    prev->next = temp->next;
    free(temp);  // Free memory
}
```

*The **time complexity** of this operation in worst case is O(N) where N is the number of nodes in the Linked List.*

- **Deleting a node at a given position**: If the node to be deleted is the root node, we can simply delete it by updating the head pointer to point to the next of the root node. To delete a node present somewhere in between, we must have the pointer to the node previous to the node to be deleted. So if the position is not zero, run a loop position-1 times and get the pointer to the previous node and follow the method discussed in the first situation above to delete the node.

Below is the implementation of this approach:

C++ Java

```
/* Given a reference (pointer to pointer) to the head of a list
   and a position, deletes the node at the given position */
void deleteNode(struct Node **head_ref, int position)
{
   // If linked list is empty
   if (*head_ref == NULL)
      return;
   // Store head node
   struct Node* temp = *head_ref;
    // If head needs to be removed
    if (position == 0)
    {
        *head_ref = temp->next;   // Change head
        free(temp);               // free old head
        return;
    }
    // Find previous node of the node to be deleted
    for (int i=0; temp!=NULL && i<position-1; i++)
         temp = temp->next;
    // If position is more than number of ndoes
    if (temp == NULL || temp->next == NULL)
         return;
// Node temp->next is the node to be deleted Store pointer to the next of node to be dele
    struct Node *next = temp->next->next;
    // Unlink the node from linked list
    free(temp->next);  // Free memory
    temp->next = next;  // Unlink the deleted node from list
}

```

*The **time complexity** of this operation in worst case is O(N) where N is the number of nodes in the Linked List.*

- **Deleting a node whose pointer is given**: In this case a pointer is given which is pointing to a particular node in the linked list and task is to delete that particular node.

  This can be done by following a similar approach as in the above two cases, by first finding the node just previous to it and updating its next pointer. The time complexity of this would be again O(N).

  However, this particular case can be solved with ***O(1)** time complexity* if the pointer to the node to be deleted is given.

  The ***efficient solution*** is to copy the data from the next node to the node to be deleted and delete the next node. Suppose the node to be deleted is ***node_ptr***, it can be deleted as:

```
// Find next node using next pointer
struct Node *temp  = node_ptr->next;

// Copy data of next node to this node
node_ptr->data  = temp->data;

// Unlink next node
node_ptr->next  = temp->next;

// Delete next node
free(temp);
```

Note: *This solution fails if the node to be deleted is the last node of the List.*