Delete middle of linked list

Given a singly linked list, delete middle of the linked list. For example, if given linked list is 1->2->3->4->5 then linked list should be modified to 1->2->4->5

If there are even nodes, then there would be two middle nodes, we need to delete the second middle element. For example, if given linked list is 1->2->3->4->5->6 then it should be modified to 1->2->3->5->6.

If the input linked list is NULL, then it should remain NULL.

If the input linked list has 1 node, then this node should be deleted and new head should be returned.

A Simple Solution is to first count number of nodes in linked list, then delete n/2'th node using the simple deletion process.

The above solution requires two traversals of linked list. We can delete middle node using one traversal. The idea is to use two pointers, slow_ptr and fast_ptr. Both pointers start from head of list. When fast_ptr reaches end, slow_ptr reaches middle. This idea is same as the one used in method 2 of this post. The additional thing in this post is to keep track of previous of middle so that we can delete middle.

Below is C++ implementation.

```
// C++ program to delete middle of a linked list
#include<bits/stdc++.h>
using namespace std;

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

// Deletes middle node and returns head of the
// modified list
struct Node* deleteMid(struct Node *head)
{
    // Base cases
    if (head == NULL)
        return NULL;
    if (head->next == NULL)
    {
        delete head;
        return NULL;
    }

    // Initialize slow and fast pointers to reach
    // middle of linked list
    struct Node *slow_ptr = head;
    struct Node *fast_ptr = head;

    // Find the middle and previous of middle.
    struct Node *prev; // To store previous of slow_ptr
    while (fast_ptr != NULL && fast_ptr->next != NULL)
    {
        fast_ptr = fast_ptr->next->next;
        prev = slow_ptr;
        slow_ptr = slow_ptr->next;
    }

    //Delete the middle node
    prev->next = slow_ptr->next;
    delete slow_ptr;

    return head;
}
```

```
// A utility function to print a given linked list
void printList(struct Node *ptr)
{
    while (ptr != NULL)
    {
        cout << ptr->data << "->";
        ptr = ptr->next;
    }
    cout << "NULL\n";
}

// Utility function to create a new node.
Node *newNode(int data)
{
    struct Node *temp = new Node;
    temp->data = data;
    temp->next = NULL;
    return temp;
}

/* Drier program to test above function*/
int main()
{
    /* Start with the empty list */
    struct Node* head = newNode(1);
    head->next = newNode(2);
    head->next->next = newNode(3);
    head->next->next->next = newNode(4);

    cout << "Gven Linked List\n";
    printList(head);

    head = deleteMid(head);

    cout << "Linked List after deletion of middle\n";
    printList(head);

    return 0;
}
```

Output :

```
Gven Linked List
1->2->3->4->NULL
Linked List after deletion of middle
1->2->4->NULL
```