**Problem Name:** Remove nth node from end list

**Topics:** Linked list, Two Pointers

**Companies:** Facebook, Microsoft, Amazon, Google, Walmart, Apple, Bloomberg, Uber

**Level:** Medium

**Language:** C++

**Problem Statement**: Given the head of a linked list, remove the nth node from the end of the list and return its head.

**Input Format:**

First line of the input contain integer n (size of list)

Second line contain n space separated integer list values.

Last line contain integer value pos representing nth node.

Ex:

5

1 2 3 4 5

1

**Output Format:** Print linked list after removing nth node

**Constraints:**

* The number of nodes in the list is sz.
* 1 <= sz <= 30
* 0 <= Node.val <= 100
* 1 <= n <= sz

**Examples:**

**Input:** head = [1,2,3,4,5], n = 2

**Output:** [1,2,3,5]

**Solution:**

**Explanation:** Simply iterate to n with other pointer and there will be two cases 1. When n < length of list and 2. When n is equal to length of list, for 2nd case simply return head->next. And for 1st case traverse until first is not NULL with second pointer and last delete that node.

**Code:**

#include <bits/stdc++.h>

using namespace std;

class ListNode

{

    public:

        int val;

        ListNode\* next;

        ListNode(int a){

            val = a;

            next = NULL;

        }

};

void insertNode(ListNode\* &head,int val) {

    ListNode\* newNode = new ListNode(val);

    if(head == NULL) {

        head = newNode;

        return;

    }

    ListNode\* temp = head;

    while(temp->next != NULL)

     temp = temp->next;

    temp->next = newNode;

    return;

}

void printList(ListNode \*node)

{

    while (node!=NULL)

    {

        cout<<node->val<<" ";

        node = node->next;

    }

}

ListNode\* removeNthFromEnd(ListNode\* head, int n) {

    ListNode \*first = head;

    ListNode \*second = head;

    while(n--)

    {

        first = first->next;

    }

    //case where n < length of linked list

    if(first)

    {

        while(first->next)

        {

            first = first->next;

            second = second->next;

        }

        ListNode \*node\_to\_delete = second->next;

        second->next = second->next->next;

        delete node\_to\_delete;

    }

    //case where n == length of linked list

    else if(!first)

    {

        ListNode \*node\_to\_delete = head;

        head = head->next;

        delete node\_to\_delete;

    }

    return head;

}

int main()

{

    ListNode\* a = NULL;

    ListNode\* res = NULL;

    int n, temp, pos;

    cin>>n;

    while(n--){

        cin>>temp;

        insertNode(a, temp);

    }

    cin>>pos;

    res = removeNthFromEnd(a, pos);

    printList(res);

    return 0;

}

**Time Complexity**: O(N)

**Space Complexity:** O(N)

**Optimized Solution:**

**Explanation:** we can simply stagger our two pointers by **n** nodes by giving the first pointer (**fast**) a head start before starting the second pointer (**slow**). Doing this will cause **slow** to reach the **n**'th node from the end at the same time that **fast** reaches the end. Since we will need access to the node *before* the target node in order to remove the target node, we can use fast.next == null as our exit condition, rather than fast == null, so that we stop one node earlier.

This will unfortunately cause a problem when n is the same as the length of the list, which would make the first node the target node, and thus make it impossible to find the node *before* the target node. If that's the case, however, we can just return head.next without needing to stitch together the two sides of the target node.

Otherwise, once we succesfully find the node *before* the target, we can then stitch it together with the node *after* the target, and then return head.

**Code:**

#include <bits/stdc++.h>

using namespace std;

class ListNode

{

    public:

        int val;

        ListNode\* next;

        ListNode(int a){

            val = a;

            next = NULL;

        }

};

void insertNode(ListNode\* &head,int val) {

    ListNode\* newNode = new ListNode(val);

    if(head == NULL) {

        head = newNode;

        return;

    }

    ListNode\* temp = head;

    while(temp->next != NULL)

     temp = temp->next;

    temp->next = newNode;

    return;

}

void printList(ListNode \*node)

{

    while (node!=NULL)

    {

        cout<<node->val<<" ";

        node = node->next;

    }

}

ListNode\* removeNthFromEnd(ListNode\* head, int n) {

    ListNode \*fast = head, \*slow = head;

    for (int i = 0; i < n; i++)

        fast = fast->next;

    if (!fast)

        return head->next;

    while (fast->next)

        fast = fast->next, slow = slow->next;

    slow->next = slow->next->next;

    return head;

}

int main()

{

    ListNode\* a = NULL;

    ListNode\* res = NULL;

    int n, temp, pos;

    cin>>n;

    while(n--){

        cin>>temp;

        insertNode(a, temp);

    }

    cin>>pos;

    res = removeNthFromEnd(a, pos);

    printList(res);

    return 0;

}

**Time Complexity**: O(N)

**Space Complexity:** O(1)