```c
#include <stdio.h>
#include <stdlib.h>

typedef struct _listnode{
    int item;
    struct _listnode *next;
} ListNode;


void printList(ListNode *cur);
ListNode * findNode(ListNode *cur, int index);
int insertNode(ListNode **ptrHead, int index, int item);

int removeNode(ListNode **ptrHead,int index);

int main()
{
    ListNode *head=NULL;
    int size =0;
    int item;
    int index;

    printf("Enter a list of numbers, terminated by any non-digit character: \n");
    while(scanf("%d",&item))
        if(insertNode(&head,size, item)) size++;
    scanf("%*s");

    printList(head);

    while(1){
        printf("Enter the index of the node to be removed: ");
        scanf("%d",&index);

        if(removeNode(&head,index))
            size--;
        else{
            printf("The node cannot be removed.\n");
            break;
        }

        printf("After the removal operation,\n");
        printList(head);
    }

    printList(head);
    return 0;
}

void printList(ListNode *cur){
    printf("Current List: ");
    while (cur != NULL){
        printf("%d ", cur->item);
        cur = cur->next;
    }
    printf("\n");
}

ListNode *findNode(ListNode* cur, int index)
{
   if (cur==NULL || index<0)
      return NULL;
   while(index>0){
      cur=cur->next;
      if (cur==NULL)
         return NULL;
      index--;
```

```c
    }
    return cur;
}

int insertNode(ListNode **ptrHead, int index, int item){
    ListNode  *pre, *newNode;
    // If empty list or inserting first node, update head pointer
    if (index == 0){
        newNode = malloc(sizeof(ListNode));
        newNode->item = item;
        newNode->next = *ptrHead;
        *ptrHead = newNode;
        return 1;
    }
    // Find the nodes before and at the target position
    // Create a new node and reconnect the links
    else if ((pre = findNode(*ptrHead, index-1)) != NULL){
        newNode = malloc(sizeof(ListNode));
        newNode->item = item;
        newNode->next = pre->next;
        pre->next = newNode;
        return 1;
    }
    return 0;
}

int removeNode(ListNode **ptrHead,int index)
{
/* Write your program code here */
    ListNode *currentNode, *previousNode;

    // This is the prof answer
    // Checking if the LinkedList is empty:
    if (*ptrHead == NULL)
        return -1;

     // Edge case: removing node at index 0
    if (index ==0){
        currentNode = *ptrHead;
        *ptrHead = currentNode -> next;
        free(currentNode);
        return 0;
    }

    // Find the nodes before and after the target position
    // Free the target node and reconnect the links
    if ((previousNode = findNode(*ptrHead, index-1))!=NULL)
    {
        if (previousNode -> next == NULL)
            return -1;
        currentNode = previousNode ->next;
        free(currentNode);
        return 0;
    }
    return -1;

}
```