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

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

typedef struct _linkedlist{
    ListNode *head;
    int size;
} LinkedList;

void printList2(LinkedList ll);
ListNode* findNode2(LinkedList ll, int index);
int insertNode2(LinkedList *ll, int index, int item);

int removeNode2(LinkedList *ll,int index);

int main()
{
    LinkedList ll;
    ll.head =NULL;
    ll.size = 0;
    int item;
    int index;

    printf("Enter a list of numbers, terminated by any non-digit character: \n");
    while(scanf("%d",&item))
    {
        if(!insertNode2(&ll,ll.size, item)) break;
    }

    scanf("%*s");

    printList2(ll);

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

        if(!removeNode2(&ll,index)){
            printf("The node cannot be removed.\n");
            break;
        }

        printf("After the removal operation,\n");
        printList2(ll);
    }

    printList2(ll);
    return 0;
}

void printList2(LinkedList ll){
    if(ll.head != NULL){
        ListNode *cur = ll.head;
        printf("Current List has %d elements: ",ll.size);

        while (cur != NULL){
            printf("%d ", cur->item);
            cur = cur->next;
        }
        printf("\n");
    }
}
```

```
67
68  ListNode* findNode2(LinkedList ll, int index)
69  {
70     if(ll.head != NULL){
71          ListNode *cur = ll.head;
72          if (cur==NULL || index<0 || index >ll.size)
73             return NULL;
74
75          while(index>0){
76              cur=cur->next;
77              if (cur==NULL)
78                  return NULL;
79              index--;
80          }
81          return cur;
82     }
83     else
84       return NULL;
85  }
86
87  int insertNode2(LinkedList *ll, int index, int item){
88      ListNode  *pre, *newNode;
89      // If empty list or inserting first node, update head pointer
90      if (index == 0){
91          newNode = malloc(sizeof(ListNode));
92          newNode->item = item;
93          newNode->next = ll->head;
94
95          ll->head = newNode;
96          ll->size++;
97          return 1;
98      }
99      // Find the nodes before and at the target position
100     // Create a new node and reconnect the links
101     else if ((pre = findNode2(*ll, index-1)) != NULL){
102         newNode = malloc(sizeof(ListNode));
103         newNode->item = item;
104         newNode->next = pre->next;
105         pre->next = newNode;
106         ll->size++;
107         return 1;
108     }
109     return 0;
110 }
111
112 int removeNode2(LinkedList *ll,int index)
113 {
114     // Model Answer Provided in tutorial:
115     ListNode *currentNode, *previousNode;
116     currentNode = ll ->head;
117
118     // Checking if the LinkedList is empty
119     if (ll->head == NULL)
120         return -1;
121
122     // Edge case: checking if the index to be removed is at the front of the list
123     if (index == 0)
124     {
125         // setting the head pointer of the linked list to point to the next node after currentNode
126         ll -> head = currentNode -> next;
127         free(currentNode);
128         // Need to shrink LinkedList size:
129         ll ->size --;
130         return 0;
131     }
132
```

```
133      if ((previousNode == findNode2(*ll, index-1))!= NULL)
134      {
135          //Not the first Node
136          if (previousNode->next == NULL)
137              return -1;
138
139          // Iterate through the LinkedList:
140          currentNode = previousNode -> next;
141
142          // When found, linking the previous node to the node after the currentNode (i.e. the jumping past
currentNode part)
143          previousNode -> next = currentNode ->next;
144          free(currentNode);
145          // Decreasing the size of the linkedlist:
146          ll ->size --;
147          return 0;
148
149      }
150      return -1;
151  }
```