```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);
void deleteList(ListNode **ptrHead);

int split(ListNode *cur,ListNode **ptrEvenList,ListNode **ptrOddList);

int main()
{
    ListNode *head=NULL;
    ListNode *oddHead = NULL;
    ListNode *evenHead = NULL;

    int size =0;
    int item;

    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");

    printf("\nBefore split() is called:\n");
    printf("The original list:\n");
    printList(head);

    split(head, &evenHead, &oddHead);

    printf("\nAfter split() was called:\n");
    printf("The original list:\n");
    printList(head);
    printf("The even list:\n");
    printList(evenHead);
    printf("The odd list:\n");
    printList(oddHead);

    if(head!=NULL)
       deleteList(&head);
    if(oddHead!=NULL)
       deleteList(&oddHead);
    if(evenHead!=NULL)
       deleteList(&evenHead);
    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--;
    }
    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;
}

void deleteList(ListNode **ptrHead){
    ListNode *cur = *ptrHead;
    ListNode *temp;
    while (cur!= NULL) {
        temp=cur->next;
        free(cur);
        cur=temp;
    }
    *ptrHead=NULL;
}


int split(ListNode *cur, ListNode **ptrEvenList,ListNode **ptrOddList)
// The above function takes in three arguments 1) the pointer to the current LinkedList, 2) the pointer to the EvenLinkedList 3) a pointer to the OddLinkedList
// Splits the linked list into two separate linked lists
// cur - Pointer to the head of the original linked list
// ptrEvenList - Pointer to the head of the even linked list
// ptrOddList - Pointer to the head of the odd linked list
{
    // Pointer to keep track of the current node in the even linked list
    ListNode *currentEven = NULL;
    // Pointer to keep track of the tail node in the even linked list
    ListNode *tailEven = NULL;

    // Pointer to keep track of the current node in the odd linked list
    ListNode *currentOdd = NULL;
    // Pointer to keep track of the tail node in the odd linked list
    ListNode *tailOdd = NULL;

    // Index of the current node in the original linked list
    int nodeIndex = 0;
    // This will keep track of the index of the current node in the original linked list

    // Iterate through the original linked list
```

```c
    while (cur != NULL) { // will return null at the end, hence as long as cur != null, the program will keep traversing


        // If the node index is even. This part is for the Even LinkedList
        if (nodeIndex % 2 == 0) {
            // If the current node in the even linked list is null, it means the list is empty
            if (currentEven == NULL) {
                // Set the head of the even linked list to be the current node
                currentEven = cur;
                // Set the tail of the even linked list to be the current node
                tailEven = currentEven;

                // If the Linked List is not null:
            } else {
                // Add the current node to the end of the even linked list
                tailEven->next = cur;
                // Update the tail of the even linked list to be the current node
                tailEven = tailEven->next;
            }

            // Now for the odd LinkedList:
        } else {
            // If the current node in the odd linked list is null, it means the list is empty
            if (currentOdd == NULL) {
                // Set the head of the odd linked list to be the current node
                currentOdd = cur;
                // Set the tail of the odd linked list to be the current node
                tailOdd = currentOdd;

                // if the odd LinkedList is not null:
            } else {
                // Add the current node to the end of the odd linked list
                tailOdd->next = cur;
                // Update the tail of the odd linked list to be the current node
                tailOdd = tailOdd->next;
            }
        }
        // Move to the next node in the original linked list
        cur = cur->next;
        // Increment the node index
        nodeIndex++;
    }

    // Set the head pointers for the even and odd linked lists
    *ptrEvenList = currentEven;
    *ptrOddList = currentOdd;

    // Set the next pointers of the tail nodes in the even and odd linked lists to be null
    tailEven->next = NULL;
    tailOdd->next = NULL;

    // Return success
    return 0;
}
```