

```

1  #include <stdio.h>
2  #include <stdlib.h>
3
4  typedef struct _listnode{
5      int item;
6      struct _listnode *next;
7  } ListNode;
8
9
10 void printList(ListNode *cur);
11 ListNode * findNode(ListNode *cur, int index);
12 int insertNode(ListNode **ptrHead, int index, int item);
13 void deleteList(ListNode **ptrHead);
14
15 int split(ListNode *cur, ListNode **ptrEvenList, ListNode **ptrOddList);
16
17 int main()
18 {
19     ListNode *head=NULL;
20     ListNode *oddHead = NULL;
21     ListNode *evenHead = NULL;
22
23     int size =0;
24     int item;
25
26     printf("Enter a list of numbers, terminated by any non-digit character: \n");
27     while(scanf("%d",&item))
28         if(insertNode(&head,size, item)) size++;
29     scanf("%*s");
30
31     printf("\nBefore split() is called:\n");
32     printf("The original list:\n");
33     printList(head);
34
35     split(head, &evenHead, &oddHead);
36
37     printf("\nAfter split() was called:\n");
38     printf("The original list:\n");
39     printList(head);
40     printf("The even list:\n");
41     printList(evenHead);
42     printf("The odd list:\n");
43     printList(oddHead);
44
45     if(head!=NULL)
46         deleteList(&head);
47     if(oddHead!=NULL)
48         deleteList(&oddHead);
49     if(evenHead!=NULL)
50         deleteList(&evenHead);
51     return 0;
52 }
53
54 void printList(ListNode *cur){
55     printf("Current List: ");
56     while (cur != NULL){
57         printf("%d ", cur->item);
58         cur = cur->next;
59     }
60     printf("\n");
61 }
62
63 ListNode *findNode(ListNode* cur, int index)
64 {
65     if (cur==NULL || index<0)
66         return NULL;

```

```

67     while(index>0){
68         cur=cur->next;
69         if (cur==NULL)
70             return NULL;
71         index--;
72     }
73     return cur;
74 }
75
76 int insertNode(ListNode **ptrHead, int index, int item){
77     ListNode *pre, *newNode;
78     // If empty list or inserting first node, update head pointer
79     if (index == 0){
80         newNode = malloc(sizeof(ListNode));
81         newNode->item = item;
82         newNode->next = *ptrHead;
83         *ptrHead = newNode;
84         return 1;
85     }
86     // Find the nodes before and at the target position
87     // Create a new node and reconnect the links
88     else if ((pre = findNode(*ptrHead, index-1)) != NULL){
89         newNode = malloc(sizeof(ListNode));
90         newNode->item = item;
91         newNode->next = pre->next;
92         pre->next = newNode;
93         return 1;
94     }
95     return 0;
96 }
97
98 void deleteList(ListNode **ptrHead){
99     ListNode *cur = *ptrHead;
100    ListNode *temp;
101    while (cur!= NULL) {
102        temp=cur->next;
103        free(cur);
104        cur=temp;
105    }
106    *ptrHead=NULL;
107 }
108
109
110 int split(ListNode *cur, ListNode **ptrEvenList, ListNode **ptrOddList)
111 {
112     int even = 1, evenSize = 0, oddSize = 0;
113     ListNode *cur = head;
114     if (cur == NULL)
115         return -1;
116     while(cur!= NULL)
117     {
118         if (even == 1)
119         {
120             insertNode(ptrEvenList, evenSize, cur->num);
121             evenSize ++;
122         }
123         else {
124             insertNode(ptrOddList, oddSize, cur->num);
125             oddSize ++;
126         }
127         cur = cur ->next;
128         even =- even;
129     }
130     return 0;
131 }

```