

```

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 // 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
112 // Splits the linked list into two separate linked lists
113 // cur - Pointer to the head of the original linked list
114 // ptrEvenList - Pointer to the head of the even linked list
115 // ptrOddList - Pointer to the head of the odd linked list
116 {
117     // Pointer to keep track of the current node in the even linked list
118     ListNode *currentEven = NULL;
119     // Pointer to keep track of the tail node in the even linked list
120     ListNode *tailEven = NULL;
121
122     // Pointer to keep track of the current node in the odd linked list
123     ListNode *currentOdd = NULL;
124     // Pointer to keep track of the tail node in the odd linked list
125     ListNode *tailOdd = NULL;
126
127     // Index of the current node in the original linked list
128     int nodeIndex = 0;
129     // This will keep track of the index of the current node in the original linked list
130
131     // Iterate through the original linked list

```

```

132     while (cur != NULL) { // will return null at the end, hence as long as cur != null, the program will
keep traversing
133
134
135     // If the node index is even. This part is for the Even LinkedList
136     if (nodeIndex % 2 == 0) {
137         // If the current node in the even linked list is null, it means the list is empty
138         if (currentEven == NULL) {
139             // Set the head of the even linked list to be the current node
140             currentEven = cur;
141             // Set the tail of the even linked list to be the current node
142             tailEven = currentEven;
143
144             // If the Linked List is not null:
145         } else {
146             // Add the current node to the end of the even linked list
147             tailEven->next = cur;
148             // Update the tail of the even linked list to be the current node
149             tailEven = tailEven->next;
150         }
151
152         // Now for the odd LinkedList:
153     } else {
154         // If the current node in the odd linked list is null, it means the list is empty
155         if (currentOdd == NULL) {
156             // Set the head of the odd linked list to be the current node
157             currentOdd = cur;
158             // Set the tail of the odd linked list to be the current node
159             tailOdd = currentOdd;
160
161             // if the odd LinkedList is not null:
162         } else {
163             // Add the current node to the end of the odd linked list
164             tailOdd->next = cur;
165             // Update the tail of the odd linked list to be the current node
166             tailOdd = tailOdd->next;
167         }
168     }
169     // Move to the next node in the original linked list
170     cur = cur->next;
171     // Increment the node index
172     nodeIndex++;
173 }
174
175 // Set the head pointers for the even and odd linked lists
176 *ptrEvenList = currentEven;
177 *ptrOddList = currentOdd;
178
179 // Set the next pointers of the tail nodes in the even and odd linked lists to be null
180 tailEven->next = NULL;
181 tailOdd->next = NULL;
182
183 // Return success
184 return 0;
185 }

```