

```

1  #include <stdio.h>
2  #include <stdlib.h>
3
4  typedef int ListElementType;
5
6  typedef struct ListNode *ListPointer;
7  typedef struct ListNode
8  {
9      ListElementType Data;
10     ListPointer Next;
11 } ListNode;
12
13 typedef enum {
14     FALSE, TRUE
15 } boolean;
16
17
18 void CreateList(ListPointer *List);
19 boolean EmptyList(ListPointer List);
20 void LinkedInsert(ListPointer *List, ListElementType Item, ListPointer PredPtr);
21 void LinkedDelete(ListPointer *List, ListPointer PredPtr);
22 void LinkedTraverse(ListPointer List);
23 void LinearSearch(ListPointer List, ListElementType Item, ListPointer *PredPtr, boolean *Found);
24 void OrderedLinearSearch(ListPointer List, ListElementType Item, ListPointer *PredPtr, boolean *Found);
25 void Inverse_List(ListPointer *List);
26
27 int main()
28 {
29     ListPointer AList, PredPtr;
30     ListElementType Item;
31     int i, n;
32
33     CreateList(&AList);
34
35     printf("Give the number of integers:");
36     scanf("%d", &n);
37
38     for(i = 0; i < n; i++)
39     {
40         printf("Give an integer:");
41         scanf("%d", &Item);
42         PredPtr = NULL;
43         LinkedInsert(&AList, Item, PredPtr);
44     }
45
46     LinkedTraverse(AList);
47
48     Inverse_List(&AList);
49
50     LinkedTraverse(AList);
51
52     return 0;
53 }
54
55 void CreateList(ListPointer *List)
56 {
57     *List = NULL;
58 }
59
60 boolean EmptyList(ListPointer List)
61 {
62     return (List==NULL);
63 }
64
65 void LinkedInsert(ListPointer *List, ListElementType Item, ListPointer PredPtr)
66 {

```

```

67     ListPointer TempPtr;
68
69     TempPtr= (ListPointer)malloc(sizeof(struct ListNode));
70     /* printf("Insert &List %p, List %p, &(*List) %p, (*List) %p, TempPtr %p\n",
71     &List, List, &(*List), (*List), TempPtr); */
72     TempPtr->Data = Item;
73     if (PredPtr==NULL) {
74         TempPtr->Next = *List;
75         *List = TempPtr;
76     }
77     else {
78         TempPtr->Next = PredPtr->Next;
79         PredPtr->Next = TempPtr;
80     }
81 }
82
83 void LinkedDelete(ListPointer *List, ListPointer PredPtr)
84 {
85     ListPointer TempPtr;
86
87     if (EmptyList(*List))
88         printf("EMPTY LIST\n");
89     else
90     {
91         if (PredPtr == NULL)
92         {
93             TempPtr = *List;
94             *List = TempPtr->Next;
95         }
96         else
97         {
98             TempPtr = PredPtr->Next;
99             PredPtr->Next = TempPtr->Next;
100         }
101         free(TempPtr);
102     }
103 }
104
105 void LinkedTraverse(ListPointer List)
106 {
107     ListPointer CurrPtr;
108
109     if (EmptyList(List))
110         printf("EMPTY LIST\n");
111     else
112     {
113         CurrPtr = List;
114         while ( CurrPtr!=NULL )
115         {
116             printf("%d ",(*CurrPtr).Data);
117             CurrPtr = CurrPtr->Next;
118         }
119         printf("\n");
120     }
121 }
122
123 void LinearSearch(ListPointer List, ListElementType Item, ListPointer *PredPtr, boolean *Found)
124 {
125     ListPointer CurrPtr;
126     boolean stop;
127
128     CurrPtr = List;
129     *PredPtr=NULL;
130     stop= FALSE;
131     while (!stop && CurrPtr!=NULL )
132     {

```

```

133         if (CurrPtr->Data==Item )
134             stop = TRUE;
135         else
136         {
137             *PredPtr = CurrPtr;
138             CurrPtr = CurrPtr->Next;
139         }
140     }
141     *Found=stop;
142 }
143
144 void OrderedLimearSearch(ListPointer List, ListElementType Item, ListPointer *PredPtr, boolean *Found)
145 {
146     ListPointer CurrPtr;
147     boolean DoneSearching;
148
149     CurrPtr = List;
150     *PredPtr = NULL;
151     DoneSearching = FALSE;
152     *Found = FALSE;
153     while (!DoneSearching && CurrPtr!=NULL )
154     {
155         if (CurrPtr->Data>=Item )
156         {
157             DoneSearching = TRUE;
158             *Found = (CurrPtr->Data==Item);
159         }
160         else
161         {
162             *PredPtr = CurrPtr;
163             CurrPtr = CurrPtr->Next;
164         }
165     }
166 }
167
168 void Inverse_List(ListPointer *List)
169 {
170     ListPointer HelpList,CurrPtr;
171
172     /*????????? ????? ?? ?????????? ?? ?????????? ??? ??? ??????*/
173     CreateList(&HelpList);
174
175     if(!EmptyList(*List))
176     {
177         /*????????????? ??? ?????? ?????? ??? ??? ?????? ??? ??? ????????????? ???????
178         ??? ??? ?? ?????????? ????? ?????????? ??????*/
179         CurrPtr=*List;
180         while(CurrPtr->Next != NULL)
181         {
182             LinkedInsert(&HelpList,CurrPtr->Data,NULL);
183             CurrPtr=CurrPtr->Next;
184         }
185         /*????????? ??? ?? ?????????? ??????????*/
186         LinkedInsert(&HelpList,CurrPtr->Data,NULL);
187     }
188     /*????????????? ??? ?????????? ????? ??????? ??????*/
189     *List=HelpList;
190 }

```