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

typedef int ListElementType;

typedef struct ListNode *ListPointer;
typedef struct ListNode
{
    ListElementType Data;
    ListPointer Next;
} ListNode;

typedef enum {
    FALSE, TRUE
} boolean;


void CreateList(ListPointer *List);
boolean EmptyList(ListPointer List);
void LinkedInsert(ListPointer *List, ListElementType Item, ListPointer PredPtr);
void LinkedDelete(ListPointer *List, ListPointer PredPtr);
void LinkedTraverse(ListPointer List);
void LinearSearch(ListPointer List, ListElementType Item, ListPointer *PredPtr, boolean *Found);
void OrderedLimearSearch(ListPointer List, ListElementType Item, ListPointer *PredPtr, boolean *Found);
void append_list_element(ListPointer *List, ListElementType Item);

int main()
{
    ListPointer AList,PredPtr;
    ListElementType Item;
    int i,n;

    CreateList(&AList);

    printf("Give the number of integers:");
    scanf("%d",&n);

    for(i = 0; i < n; i++)
    {
        printf("Give an integer:");
        scanf("%d",&Item);
        PredPtr = NULL;
        LinkedInsert(&AList,Item,PredPtr);
    }

    LinkedTraverse(AList);

    printf("\nGive the integer you want to insert at the end of the list:");
    scanf("%d",&Item);

    append_list_element(&AList,Item);

    LinkedTraverse(AList);

    return 0;
}

void CreateList(ListPointer *List)
{
    *List = NULL;
}

boolean EmptyList(ListPointer List)
{
    return (List==NULL);
}
```

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

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