```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 Intersection_List(ListPointer ListA,ListPointer ListB,ListPointer *FinalList);

int main()
{
    ListPointer List_A,List_B,List_Intersection;
    ListElementType Item;
    int i,n;

    CreateList(&List_A);
    CreateList(&List_B);

    /*???????? ??????? ????????? ??? ??? ? ?????.*/
    printf("Give the number of integers for List A:");
    scanf("%d",&n);

    /*???????? ????????? ??? ????????.*/
    printf("----List_A----\n");
    for(i = 0; i < n; i++)
    {
        printf("Give an integer:");
        scanf("%d",&Item);
        LinkedInsert(&List_A,Item,NULL);
    }

    /*???????? ??????? ????????? ??? ??? ? ?????.*/
    printf("\nGive the number of integers for List B:");
    scanf("%d",&n);

    /*???????? ????????? ??? ????????.*/
    printf("----List_B----\n");
    for(i = 0; i < n; i++)
    {
        printf("Give an integer:");
        scanf("%d",&Item);
        LinkedInsert(&List_B,Item,NULL);
    }

    /*???????? ? ??????.*/
    printf("\n----List_A----\n");
    LinkedTraverse(List_A);

    /*???????? ? ??????.*/
    printf("\n----List_B----\n");
```

```
    LinkedTraverse(List_B);
    /*????? ?????????? ??? ?????? ?????.*/
    Intersection_List(List_A,List_B,&List_Intersection);

    /*???????? ??????? ?????? ??? ???????? ??? ????.*/
    printf("\n----Intersection List----\n");
    LinkedTraverse(List_Intersection);

    return 0;
}

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

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

void LinkedInsert(ListPointer *List, ListElementType Item, ListPointer PredPtr)
{
    ListPointer TempPtr;

   TempPtr= (ListPointer)malloc(sizeof(struct ListNode));
 /*  printf("Insert &List %p, List %p,  &(*List) %p, (*List) %p, TempPtr %p\n",
   &List, List,  &(*List), (*List), TempPtr); */
   TempPtr->Data = Item;
    if (PredPtr==NULL) {
        TempPtr->Next = *List;
        *List = TempPtr;
    }
    else {
        TempPtr->Next = PredPtr->Next;
        PredPtr->Next = TempPtr;
    }
}

void LinkedDelete(ListPointer *List, ListPointer PredPtr)
{
    ListPointer TempPtr;

    if (EmptyList(*List))
        printf("EMPTY LIST\n");
   else
   {
        if (PredPtr == NULL)
        {
              TempPtr = *List;
              *List = TempPtr->Next;
        }
        else
        {
             TempPtr = PredPtr->Next;
             PredPtr->Next = TempPtr->Next;
        }
        free(TempPtr);
    }
}

void LinkedTraverse(ListPointer List)
{
    ListPointer CurrPtr;

   if (EmptyList(List))
```

```
133         printf("EMPTY LIST\n");
134    else
135    {
136         CurrPtr = List;
137          while ( CurrPtr!=NULL )
138         {
139              printf("%d ",(*CurrPtr).Data);
140              CurrPtr = CurrPtr->Next;
141         }
142         printf("\n");
143    }
144 }
145
146 void LinearSearch(ListPointer List, ListElementType Item, ListPointer *PredPtr, boolean *Found)
147 {
148    ListPointer CurrPtr;
149    boolean stop;
150
151    CurrPtr = List;
152     *PredPtr=NULL;
153    stop= FALSE;
154    while (!stop && CurrPtr!=NULL )
155     {
156          if (CurrPtr->Data==Item )
157             stop = TRUE;
158          else
159          {
160             *PredPtr = CurrPtr;
161             CurrPtr = CurrPtr->Next;
162          }
163     }
164     *Found=stop;
165 }
166
167 void OrderedLimearSearch(ListPointer List, ListElementType Item, ListPointer *PredPtr, boolean *Found)
168 {
169    ListPointer CurrPtr;
170    boolean DoneSearching;
171
172    CurrPtr = List;
173    *PredPtr = NULL;
174    DoneSearching = FALSE;
175    *Found = FALSE;
176    while (!DoneSearching && CurrPtr!=NULL )
177     {
178          if (CurrPtr->Data>=Item )
179          {
180             DoneSearching = TRUE;
181             *Found = (CurrPtr->Data==Item);
182          }
183          else
184          {
185             *PredPtr = CurrPtr;
186             CurrPtr = CurrPtr->Next;
187          }
188     }
189 }
190
191 void Intersection_List(ListPointer ListA,ListPointer ListB,ListPointer *FinalList)
192 {
193     ListPointer TempPtr1,TempPtr2;
194
195     /*?????????? ??? ??????? ????? ??? ?? ???????? ??? ????.*/
196     CreateList(FinalList);
197
198     TempPtr1 = ListA;
```

```
199
200      while(TempPtr1 != NULL)
201      {
202          TempPtr2 = ListB;
203
204          while(TempPtr2 != NULL)
205          {
206              if(TempPtr1->Data == TempPtr2->Data)
207              {
208                  LinkedInsert(FinalList,TempPtr1->Data,NULL);
209                  break;
210              }
211
212
213              TempPtr2=TempPtr2->Next;
214          }
215
216          TempPtr1 = TempPtr1->Next;
217      }
218
219  }
```