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

#define NumberOfNodes 20
#define NilValue -1

typedef int ListElementType;
typedef int ListPointer;

typedef struct {
    ListElementType Data;
    ListPointer  Next;
} NodeType;

typedef enum {
    FALSE, TRUE
} boolean;

void InitializeStoragePool(NodeType Node[], ListPointer *FreePtr);
void CreateLList(ListPointer *List);
boolean EmptyLList(ListPointer List);
boolean FullLList(ListPointer FreePtr);
void GetNode(ListPointer *P, ListPointer *FreePtr, NodeType Node[]);
void ReleaseNode(NodeType Node[NumberOfNodes], ListPointer P, ListPointer *FreePtr);
void Insert(ListPointer *List, NodeType Node[],ListPointer *FreePtr, ListPointer PredPtr, ListElementType
Item);
void Delete(ListPointer *List, NodeType Node[], ListPointer *FreePtr, ListPointer PredPtr);
void TraverseLinked(ListPointer List, NodeType Node[]);
void sort_list(ListPointer *List, NodeType Node[], boolean Ascending);
void swap(ListPointer PrevPtr,ListPointer CurrentPtr,ListPointer NextPtr,NodeType Node[],ListPointer *List
);
int Menu();

int main()
{
    ListPointer AList,FreePtr,PredPtr;
    NodeType Node[NumberOfNodes];
    ListElementType AnItem,numb,i;
    boolean asc;

    InitializeStoragePool(Node, &FreePtr);
    CreateLList(&AList);

    /*??????? ??? ?????? ??? ???????? ??? ??????? ?? ?????????
    ??? ????? ??? ????????? ?????? ??????????.*/
    do{
        printf("Enter number of integers: ");
        scanf("%d",&numb);
        if(numb < 0 || numb >20)
            printf("MH EPITREPTOS ARI8MOS.PROSPA8ISTE KSANA.\n");
    }while(numb < 0 || numb >20);

    /*???????? ??? ??????? ???? ?????.*/
    for(i=0; i < numb; i++)
    {
        printf("Enter an integer: ");
        scanf("%d",&AnItem);
        PredPtr=NilValue;
        Insert(&AList,Node,&FreePtr,PredPtr,AnItem);
    }

    /*???????? ??? Menou ??? ????????? ??? ??????? ?????????? asc
    ??????? ?? ??? ??????? ??? ??????.*/

    asc=TRUE;

```

```
65      if(Menu()==2)
66          asc=FALSE;
67
68      /*????? ?????????? ??????????.*/
69      sort_list(&AList,Node,asc);
70
71      /*???????? ???????????? ??????.*/
72      TraverseLinked(AList,Node);
73
74      return 0;
75  }
76
77  void InitializeStoragePool(NodeType Node[], ListPointer *FreePtr)
78  {
79     int i;
80
81      for (i=0; i<NumberOfNodes-1;i++)
82      {
83          Node[i].Next=i+1;
84          Node[i].Data=-1;
85      }
86      Node[NumberOfNodes-1].Next=NilValue;
87      Node[NumberOfNodes-1].Data=NilValue;
88      *FreePtr=0;
89  }
90
91  void CreateLList(ListPointer *List)
92  {
93    *List=NilValue;
94  }
95
96  boolean EmptyLList(ListPointer List)
97  {
98    return (List==NilValue);
99  }
100
101 boolean FullLList(ListPointer FreePtr)
102 {
103   return (FreePtr == NilValue);
104 }
105
106 void GetNode(ListPointer *P, ListPointer *FreePtr, NodeType Node[])
107 {
108   *P = *FreePtr;
109   if (!FullLList(*FreePtr))
110     *FreePtr =Node[*FreePtr].Next;
111 }
112
113 void ReleaseNode(NodeType Node[], ListPointer P, ListPointer *FreePtr)
114 {
115   Node[P].Next =*FreePtr;
116   Node[P].Data = -1;
117
118   *FreePtr =P;
119 }
120
121 void Insert(ListPointer *List, NodeType Node[],ListPointer *FreePtr, ListPointer PredPtr, ListElementType
Item)
122 {
123   ListPointer TempPtr;
124   GetNode(&TempPtr,FreePtr,Node);
125   if (!FullLList(TempPtr)) {
126     if (PredPtr==NilValue)
127     {
128         Node[TempPtr].Data =Item;
129         Node[TempPtr].Next =*List;
```

```
        *List =TempPtr;
    }
    else
      {
        Node[TempPtr].Data =Item;
        Node[TempPtr].Next =Node[PredPtr].Next;
        Node[PredPtr].Next =TempPtr;
      }
}
  else
    printf("Full List ...\n");
}

void Delete(ListPointer *List, NodeType Node[], ListPointer *FreePtr, ListPointer PredPtr)
{
  ListPointer TempPtr ;

  if (!EmptyLList(*List))
    if (PredPtr == NilValue)
    {
        TempPtr =*List;
        *List =Node[TempPtr].Next;
        ReleaseNode(Node,TempPtr,FreePtr);
    }
    else
      {
        TempPtr =Node[PredPtr].Next;
        Node[PredPtr].Next =Node[TempPtr].Next;
        ReleaseNode(Node,TempPtr,FreePtr);
      }
  else
    printf("Empty List ...\n");
}

void TraverseLinked(ListPointer List, NodeType Node[])
{
  ListPointer CurrPtr;

  if (!EmptyLList(List))
  {
      CurrPtr =List;
      while (CurrPtr != NilValue)
      {
          printf("%d ",Node[CurrPtr].Data);
          CurrPtr=Node[CurrPtr].Next;
       }
       printf("\n");
   }
  else printf("Empty List ...\n");
}


int Menu()
{
    int answer;

    printf("----------------------- Select ----------------------\n");
    printf("1.Sort Ascending\n");
    printf("2.Sort Descending\n");

    do{
        scanf("%d",&answer);

        if(answer!=1 && answer!=2)
            printf("WRONG!Pleas try again.");

```

```
196     }while(answer!=1 && answer!=2);
197
198     return answer;
199
200 }
201
202 //? ????????? ?????????? ???? ????? ??? ??????.
203 void sort_list(ListPointer *List, NodeType Node[], boolean Ascending)
204 {
205     ListPointer NextPtr,CurrPtr,PrevPtr,OutPtr;
206     boolean swi=FALSE;
207
208     if(!EmptyLList(*List))
209     {
210         OutPtr = *List;
211
212         while(OutPtr != NilValue)
213         {
214             CurrPtr = *List;
215             PrevPtr = NilValue;
216             NextPtr = Node[CurrPtr].Next;
217
218             while(NextPtr != NilValue)
219             {
220                 swi = FALSE;
221
222                 if(Ascending == TRUE)
223                 {
224                     if(Node[CurrPtr].Data > Node[NextPtr].Data)
225                     {
226                         swi = TRUE;
227                     }
228                 }
229                 else
230                 {
231                     if(Node[CurrPtr].Data < Node[NextPtr].Data)
232                     {
233                         swi = TRUE;
234                     }
235                 }
236                 if(swi == TRUE)
237                 {
238                     swap(PrevPtr,CurrPtr,NextPtr,Node,&(*List));
239
240                     if(OutPtr==CurrPtr)
241                     {
242                         OutPtr = NextPtr;
243                     }
244
245                 }
246                 PrevPtr=CurrPtr;
247                 CurrPtr=NextPtr;
248                 NextPtr=Node[CurrPtr].Next;
249             }
250             OutPtr--;
251         }
252     }
253     else
254     {
255         printf("Empty List...\n");
256     }
257 }
258
259 //? ????????? ????????? ???? ????? ??? ??????.
260 void swap(ListPointer PrevPtr,ListPointer CurrentPtr,ListPointer NextPtr,NodeType Node[],ListPointer *List)
261 {
```

```
262
263
264      if(CurrentPtr == *List )
265      {
266          *List = NextPtr;
267      }
268
269      Node[CurrentPtr].Next= Node[NextPtr].Next;
270      Node[NextPtr].Next = CurrentPtr;
271
272      if(PrevPtr != NilValue)
273      {
274          Node[PrevPtr].Next = NextPtr;
275      }
276  }
277
278
```