```c
/*****************************************************************************/
/*                                                                           */
/*    Doppelt verkettete sortierte Liste                                     */
/*                                                                           */
/*    (C) by Daniel Becker, letzte Änderung 04.05.99                         */
/*                                                                           */
/*****************************************************************************/
#include <stdlib.h>
#include <alloc.h>

struct liste
{
   int key;
   struct liste *next, *vor;
} *wurzel=NULL;

void insert_liste(int key)
{
   struct liste *new,*lauf;

   new=(struct liste *)malloc(sizeof(struct liste));
   new->key=key; new->next=new->vor=NULL;

   if (!wurzel) wurzel=new;
   else
   {
      for (lauf=wurzel; lauf->next && lauf->key<key; lauf=lauf->next);

      if (lauf->key==key) /* Gleiches Element */
         lauf->key=key;
      else
      if (lauf->key<key) /* Bin am Ende der Liste, lauf->next=NULL */
      {
         lauf->next=new; new->vor=lauf; /* Hinten anfügen */
      }
      else
      {
         new->next=lauf;     /* News naechster ist lauf */
         new->vor=lauf->vor; /* News vorgaenger ist der ehemalige von lauf */

         if (new->vor)       /* Der Vorgaenger (falls existiert) zeigt auf new */
            new->vor->next=new;

         lauf->vor=new;      /* Laufs vorgaenger ist ebenfalls new */

         if (lauf==wurzel)   /* Wurzelzeiger umhaengen, falls noetig */
            wurzel=new;
      }
   }
}

void delete_liste(int key)
{
   struct liste *lauf;

   if (!wurzel) return; /* Leere Listen braucht man nicht zu durchsuchen */

   for (lauf=wurzel; lauf->next && lauf->key!=key; lauf=lauf->next);
   if (lauf->key==key) /* tatsächlich gefunden... */
   {
    /* rausbrechen, also laufs vorgaenger muss auf laufs nachfolger zeigen
       und laufs nachfolger zeigt auf laufs vorgaenger, falls es sie gibt */
      if (lauf->vor) lauf->vor->next=lauf->next;
      if (lauf->next) lauf->next->vor=lauf->vor;
      if (wurzel==lauf) wurzel=lauf->next; /* Wurzel umhaengen. */
      free(lauf);
   }
}

struct liste *find_element(int key)
{
   struct liste *lauf;
```

```
   /* Suche von der Wurzel aus bis zum letzten Element oder zu dem
      richtigen. Gib den Zeiger zurück... */
   for (lauf=wurzel; lauf && lauf->key!=key; lauf=lauf->next);

   return(lauf); /* NULL bei nicht gefunden, ansonsten OK */
}


/* Gibt die gesamte Liste wieder frei... */
void freelist()
{
   struct liste *help,*h;

   help=wurzel;
   while (help)
   {
      h=help->next;
      free(help);
      help=h;
   }
}

/*****************************************************************************/
/*   Testroutine                                                             */
/*****************************************************************************/

void ausgabe()
{
   struct liste *help;

   help=wurzel;
   while (help)
   {
      printf("%d ",help->key);
      help=help->next;
   }
}

void main()
{
   int test[]={3,2,5,9,7,1,4,8,6,10},i;

   for (i=0; i<10; i++)
   {
      insert_liste(test[i]);
      clrscr();
      ausgabe();
      bioskey(0);
   }

   printf("Zeiger von 10: %p (Element: %d)",find_element(10),(struct liste *)find_element(10
   )->key);
   bioskey(0);

   for (i=0; i<10; i++)
   {
      delete_liste(test[i]);
      clrscr();
      ausgabe();
      bioskey(0);
   }

   freelist(); /* Eigentlich nicht noetig, da Liste schon leer. */
}SUB
```