```
/*****************************************************************************/
/*                                                                           */
/*    Binäre Bäume (ausgeglichen)                                            */
/*                                                                           */
/*    (C) by Daniel Becker, letzte Änderung 03.05.99                         */
/*                                                                           */
/*    Lehrbuch : Niklaus Wirth, Alorithmen und Datenstrukturen, 4. Auflage */
/*****************************************************************************/

#include <stdlib.h>
#include <math.h> /* Nur fuer die Ausgabe noetig... */

struct baum
{
   struct baum *links, *rechts;
   int hdiff; /* -1 - +1 oder -2,+2 bei Unausgeglichenheit */
   int schluessel; /* Schluesselelement, kann ausgeweitet werden */
} *b_wurzel=NULL;

struct baum *insert(int el)
{
   struct baum *b_new;

   b_new=(struct baum *)malloc(sizeof(struct baum));
   if (!b_new)
   {
      puts("Kein Speicher!");
      exit(3);
   }
   b_new->rechts=b_new->links=NULL; /* Noch keiner folgenden */
   b_new->hdiff=0; /* Noch ausgewogen */
   b_new->schluessel=el;
   return(b_new);
}

int b_insert(int el, struct baum *vorgaenger, struct baum *b)
{
   int mehr;
   struct baum *b1,*b2;

   if (el>b->schluessel)
   {
      if (b->rechts)
      {
         mehr=b_insert(el,b,b->rechts);
      }
      else
      {
         b->rechts=insert(el); mehr=1;
      }
      if (mehr)
      {  /* rechts wurde der Ast schwerer */
         switch (b->hdiff)
         {
            case -1: b->hdiff=0; mehr=0; break;
            case  0: b->hdiff=+1; break;
            case +1: /* Ausgleichen: */
                     b1=b->rechts;
                     if (b1->hdiff == +1)
                     {
                        b->rechts=b1->links;
                        b1->links=b;
                        b->hdiff=b1->hdiff=0;
                        if (!vorgaenger)
                           b_wurzel=b1;
                        else
                           vorgaenger->rechts=b1;
                     }
                     else
                     {
                        b2=b1->links;
                        b1->links=b2->rechts; b2->rechts=b1;
                        b->rechts=b2->links; b2->links=b;
```

```
                        b2->hdiff=0;
                        if (b2->hdiff == +1) b->hdiff=-1; else b->hdiff=0;
                        if (b2->hdiff == -1) b1->hdiff=+1; else b1->hdiff=0;
                        if (!vorgaenger)
                           b_wurzel=b2;
                        else
                           vorgaenger->rechts=b2;
                     }
                     mehr=0; break;
         }
      }
   }
   else
   if (el<b->schluessel)
   {
      /* Analog, wie oben! */
      if (b->links)
      {
         mehr=b_insert(el,b,b->links);
      }
      else
      {
         b->links=insert(el); mehr=1;
      }
      if (mehr)
      {  /* links wurde der Ast schwerer */
         switch (b->hdiff)
         {
            case +1: b->hdiff=0; mehr=0; break;
            case  0: b->hdiff=-1; break;
            case -1: /* Ausgleichen: */
                     b1=b->links;
                     if (b1->hdiff == -1)
                     {
                        b->links=b1->rechts;
                        b1->rechts=b;
                        b->hdiff=b1->hdiff=0;
                        if (!vorgaenger)
                           b_wurzel=b1;
                        else
                           vorgaenger->links=b1;
                     }
                     else
                     {
                        b2=b1->rechts;
                        b1->rechts=b2->links; b2->links=b1;
                        b->links=b2->rechts; b2->rechts=b;
                        b2->hdiff=0;
                        if (b2->hdiff == -1) b->hdiff=+1; else b->hdiff=0;
                        if (b2->hdiff == +1) b1->hdiff=-1; else b1->hdiff=0;
                        if (!vorgaenger)
                           b_wurzel=b2;
                        else
                           vorgaenger->links=b2;

                     }
                     mehr=0; break;
         }
      }
   }
   else
   {
      b->schluessel=el; /* gefunden, überschreiben... */
      mehr=0;
   }
   return(mehr);
}

void insert_element(int el) /* User-Funktion */
{
   if (!b_wurzel)
      b_wurzel=insert(el); /* Leerer Baum, Wurzel füllen */
   else
```

```
      b_insert(el,NULL,b_wurzel); /* Vorgänger==NULL, da Wurzel!!! */
}

void eleminiere_baum(struct baum *b)
{
   if (b->rechts) eleminiere_baum(b->rechts);
   if (b->links) eleminiere_baum(b->links);
   free(b);
}

void print_sorted(struct baum *b)
{
   if (b->links) print_sorted(b->links);
   printf("%d[%d] ",b->schluessel, b->hdiff);
   if (b->rechts) print_sorted(b->rechts);
}

/*************************************************************************/
/* Testroutinen */

void print_baum(struct baum *b, int rek_depth, int pos,int to)
{
   int x,y;

   y=rek_depth+1;
   x=pos+(40/pow(2,rek_depth)*to);

   if (b->rechts) print_baum(b->rechts, rek_depth+1, x, +1);
   if (b->links) print_baum(b->links, rek_depth+1, x, -1);

   gotoxy(x,y);
   printf("%d[%d]",b->schluessel,b->hdiff);
}

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

   for (i=0; i<10; i++)
   {
      insert_element(test[i]);
      clrscr();
      print_baum(b_wurzel,0,0,1);
      bioskey(0);
   }
   clrscr();
   print_sorted(b_wurzel);
   bioskey(0);
   clrscr();
   eleminiere_baum(b_wurzel);
}SUB
```