Programmierübungen Informatik 2a im WS 2003/04

Aufgabe 1:

Schreiben Sie eine Funktion ausschnitt, die aus einer Zeichenkette einen Teilstring extrahiert und den ausgeschnittenen Teil in eine zweite Zeichenkette kopiert. Der Ausschnitt wird durch Angabe des ersten zu kopierenden Zeichens (z.B. das Zeichen 'a') und der Anzahl der zu kopierenden Zeichen definiert.

Loesen Sie diese Aufgabe zunaechst unter ausschliesslicher Benutzung von Vektoren (Feldern) und in einer zweiten Version unter Benutzung von Zeigern.

Erstellen sie ebenfalls ein Hauptprogramm, das die implementierten Funktionen durch geeignete Aufrufe testet. Das Hauptprogramm und die entwickelten Funktionen sollen sich dabei gemeinsam in einer Datei (z.B. AUFGABE1.C) befinden.

Aufgabe 2:

Zeigervariablen werden häufig zur Realisierung dynamischer Variablen und verketteter Datenstrukturen verwendet. Zum Beispiel kann im wichtigen Fall einer einfach verketteten Liste ein Listenelement durch einen Datentyp, bestehend aus einem "Datenteil" und einem Zeiger "Naechstes" auf das nachfolgende Listenelement, repräsentiert werden.

Das Ende einer Liste wird durch den Zeigerwert NULL in der Komponente Naechstes des letzten Listenelementes gekennzeichnet. Den Listenanfang enthält eine Zeigervariable, die auf das erste Listenelement "zeigt".

Die an den Listenanfang oder an das Listenende einer verketteten Liste ein- bzw. anzufügenden Listenelemente werden dynamisch erzeugt (malloc) und, falls sie nicht mehr benötigt werden, wird der durch sie belegte Speicherplatz wieder freigegeben (free).

a) Ergänzen Sie das zu dieser Aufgabenstellung in der Datei MAIN_1.C bereitgestellte Hauptprogramm um geeignete nichtrekursive Realisierungen der Funktionen Einfuege (an den Listenanfang), Anfuegen (an das Listenende), Loeschen (eines Listenelements), ListeAnzeigen und ListeLoeschen. Speichern Sie das so ergänzte Programm unter dem Namen MAIN_1A.C ab.

b) Lagern Sie die unter a) realisierten Funktionen in eine Datei (ein eigenes Modul) mit dem Namen EVLISTE.C aus, so dass sie vom Hauptprogramm getrennt übersetzt werden können. Lagern Sie des weiteren die Prototypen der Funktionen, die von diesen evtl. benötigten #define-Konstanten und die Datentypen in die Datei EVLISTE.H aus. Ändern Sie dann das beigefügte Hauptprogramm so ab, dass es die Vereinbarungen aus EVLISTE.H benutzt. Legen Sie das so geänderte Hauptprogramm in der Datei MAIN_1B.C ab und binden Sie es mit den von Ihnen in EVLISTE.C hinterlegten Funktionen zu einem lauffähigen Programm zusammen.

c) Ergänzen Sie Ihre Lösung zur Aufgabe 1-a) so, dass die von Ihnen zu

implementierenden Funktionen Anfuegen (an das Listenende), Loeschen, ListeAnzeigen und ListeLoeschen auch in rekursiver Form vorliegen. Lagern Sie die rekursiv realisierten Routinen zusammen mit einer nichtrekursiven Implementierung der Funktion Einfuegen, fuer die eine rekursive Loesung nicht sinnvoll ist, in die Datei EVLISTER.C aus und binden Sie die Funktionen mit der Datei MAIN_1B.C zu einem lauffähigen Programm zusammen.

Aufgabe 3:

Schreiben Sie ein Programm, das von der Tastatur eingegebene Strings der maximalen Länge 20 Zeichen mit Hilfe eines binären Baumes aufsteigend sortiert und nach Eingabe des Strings "#" sortiert ausgibt; dabei sollen, da Großbuchstaben vor dem Einsortieren in Kleinbuchstaben zu konvertieren sind, die Groß-/Kleinschreibung bei der Sortierung unberücksichtigt bleiben und die Buchstaben 'ä', 'ö', 'ü' und ' ßwie "ae", "oe", "ue" bzw. "ss" einsortiert werden. Die Strings selbst sollen nicht konvertiert werden.

Neben dem sortierten Einfuegen von Strings in den binaeren Baum und dem Ausgeben des sortierten Baums (BaumAusgeben) soll auch das Loeschen des sortierten Baums durch eine Funktion BaumLoeschen unterstuetzt werden. Mit der Funktion BaumLoeschen wird der beim Aufbau des Baumes dynamisch allokierte Speicherplatz (malloc) wieder freigegeben.

Hinterlegen Sie die zu implementierenden Funktionen Einfuegen, BaumAusgeben und BaumLoeschen in der Datei SORTBAUM.C, die Prototypen dieser Funktionen zusammen mit den benötigten Datentypen in der Datei SORTBAUM.H und eine geeignete Implementierung der Funktion main in der Datei MAIN_2.C. Binden Sie SORTBAUM.C und MAIN_2.C zu einem lauffähigen Programm zusammen.

Berücksichtigen Sie bei Ihrer Lösung, daß die Operation des Einfügens von Strings in den sortierten Baum aufgrund von fehlendem dynamischen Speicher auch scheitern kann. In diesem Fall soll eine entsprechende Fehlermeldung erfolgen, der bis dahin aufgebaute Baum am Bildschirm ausgegeben und vor dem Beenden der von ihm belegte Speicherplatz wieder freigegeben werden.

Lösungen:

Aufgabe 1:

```
/*Aufgabe 1_a Lösung mit Vektoren*/

# include <stdio.h>
# include <string.h>

# define SIZE 51

int ausschnitt (char s[], char t[], char z, int x);

int main (void)
{
        char s[SIZE];
        char t[SIZE];
        char z;
        int x=0;
        int a=0;
        int b=0;
        int c=0;

        for (c=0; c<=SIZE; c++)
        {
                s[c]=t[c]=0;
        }


        printf("\nGeben Sie den String s ein, max. Zeichen 50:\t");
        scanf ("%s", &s);

        printf ("\nGeben Sie den Buchstaben ein, ab den kopiert werden soll:\t");
        scanf ("%s", &z);

        printf ("\nGeben Sie die Anzahl der Zeichen ein, die kopiert werden sollen:\t");
        scanf ("%d", &x);


        a = strlen (s);


        if (a >= SIZE)
        {
                printf ("\nDer eingegebene String ist zu lang! Max. 50 Zeichen\n\n");
                return 0;
        }
        else if (x >= a)
        {
                printf ("\nDer String eingegebene String hat nur %d Zeichen. Kopieren
nicht moeglich!\n\n", a);
                return 0;
        }


        b = ausschnitt (s, t, z, x);

        switch (b)
        {
        case 0:
                printf ("\nFEHLER!!!\n\n");

        case 1:
                printf ("\nString t lautet:\t %s\n\n", t);
                break;

        case 2:
                printf ("\nString s enthaelt nicht das Zeichen\t%c\n\n", z);
                break;
```

```
        case 3:
                printf ("\nReststring von s nach t kopiert:\t%s\n\n", t);
                break;

        default:
                break;
        }

                /*printf ("\nDer eingegebene String hat %d Zeichen. Kopieren moeglich! %
d Zeichen sollen kopiert werden.\n Der String lautet: %s\nAb Zeichen %c soll kopiert
werden.\n\n", a, x, s, z);*/
                return 0;

}



int ausschnitt ( char s[], char t[], char z, int x )
{

        int i=0;
        int j=0;
        int k=0;


        while ( s[i] != z  &&  s[i] != '\0' )
        {
                i++;
        }


        if ( s[i] == z )
        {

                while ( j<x && s[i]!='\0' )
                {
                        t[j] = s[i];
                        i++;
                        j++;

                        if ( s[i] != '\0' )
                        {
                                k++;
                        }

                }


                if ( (x>k) && (k==0) )
                        return 2;

                else if ( (x>k) && (k!=0) )
                        return 2;

                else
                        return 1;

        }

        if ( s[i] == '\0' )
                return 2;

        return 0;
}
```

/*Aufgabe 1_b Lösung mit Zeigern*/

```
# include <stdio.h>
/*# include <string.h>*/

# define SIZE 51

int ausschnitt (char *s, char *t, char z, int x);

int main (void)
{
        char s[SIZE];
        char t[SIZE];
        char z;
        int x=0;
        int a=0;
        int b=0;
        int c=0;

        for (c=0; c<=SIZE; c++)
        {
                *(s+c) = *(t+c) =0;
        }


        printf("\nGeben Sie den String s ein, max. Zeichen 50:\t");
        for ( c=0; (*(s+c)=getchar()) != '\n'; c++)
                ;

        printf ("\nGeben Sie den Buchstaben ein, ab den kopiert werden soll:\t");
        z = getchar();

        printf ("\nGeben Sie die Anzahl der Zeichen ein, die kopiert werden sollen:\t");
        scanf ("%d", &x);


        if (c >= SIZE)
        {
                printf ("\nDer eingegebene String ist zu lang! Max. 50 Zeichen\n\n");
                return 0;
        }

        else if (x >= c)
        {
                printf ("\nDer String eingegebene String hat nur %d Zeichen. Kopieren
nicht moeglich!\n\n", c);
                return 0;
        }


        b = ausschnitt (s, t, z, x);

        switch (b)
        {
        case 0:
                printf ("\nFEHLER!!!\n\n");

        case 1:
                printf ("\nString t lautet:\t %s\n\n", t);
                break;

        case 2:
                printf ("\nString s enthaelt nicht das Zeichen\t%c\n\n", z);
                break;

        default:
                break;
        }

                printf ("\nDer eingegebene String hat %d Zeichen. Kopieren moeglich! %d
Zeichen sollen kopiert werden.\n Der String lautet: %s\nAb Zeichen %c soll kopiert
```

```
werden.\n\n", a, x, s, z);
                return 0;

}



int ausschnitt ( char (*s), char (*t), char z, int x )
{

        int i=0;
        int j=0;
        int k=0;


        while ( *(s+i) != z  &&  *(s+i) != '\0' )
        {
                i++;
        }


        if ( *(s+i) == z )
        {

                while ( j<x && *(s+i)!='\0' )
                {
                        *(t+j) = *(s+i);
                        i++;
                        j++;

                        if ( *(s+i) != '\0' )
                        {
                                k++;
                        }

                }


                if ( (x>k) && (k==0) )
                        return 2;

                else if ( (x>k) && (k!=0) )
                        return 2;

                else
                        return 1;

        }

        if ( *(s+i) == '\0' )
                return 2;

        return 0;
}
```

Aufgabe 2:

Teilaufgabe a:

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

# define MAXSTRLEN  5
# define FMTSTR     "%5s"
# define OK         0
# define HEAPERR   (-1)

typedef char STRING [MAXSTRLEN + 1];

typedef struct ListenElement
{
   STRING Datenteil;
   struct ListenElement *Naechstes;
}
ListenElement;


/*Listenelemente am Ende der Liste anfügen*/

int Anfuegen(ListenElement **ppListe, STRING Wert)
{
  int i;
  ListenElement *tmp;
  ListenElement *NEW;

  NEW = (ListenElement*) malloc(sizeof(ListenElement));

  if (NEW == NULL)
         return HEAPERR;

  if(*ppListe == NULL)
         {
           *ppListe = NEW;
           (**ppListe).Naechstes = NULL;
         }

  tmp = *ppListe;

  while ((*tmp).Naechstes != NULL )
         tmp = (*tmp).Naechstes;

  for (i=0; i<=MAXSTRLEN; ++i)

  *((*NEW).Datenteil+i) = *(Wert+i);
  (*NEW).Naechstes = NULL;

  if( ! (tmp==NEW))
         (*tmp).Naechstes=NEW;

return OK;
}


/*Listemelemente am Anfang der Liste einfügen*/

int Einfuegen(ListenElement **ppListe, STRING Wert)
{
  int i;
  ListenElement *NEW;

  NEW = (ListenElement*) malloc(sizeof(ListenElement));

  if (NEW == NULL)
         return HEAPERR;
```

```
  for (i=0; i<=MAXSTRLEN; ++i)
        *(NEW->Datenteil+i) = *(Wert+i);
        NEW->Naechstes = (*ppListe);

        *ppListe=&(*NEW);

        return OK;
}


/*Lisenelemente löschen*/

void Loeschen(ListenElement **ppListe, STRING Wert)
{
  ListenElement *tmpv = NULL;
  ListenElement *tmp = NULL;

  if (*ppListe == NULL)
        goto ENDE;

  if (strcmp((**ppListe).Datenteil,Wert) == 0)
  {
        if ((**ppListe).Naechstes == NULL)
        {
                free(*ppListe);
                *ppListe=NULL;
                        goto ENDE;
        }

  else
  {
        tmp=*ppListe;
        *ppListe=&(*(**ppListe).Naechstes);
        free(tmp);
                goto ENDE;
  }

  }

  if ((**ppListe).Naechstes==NULL)
        goto ENDE;

  tmpv = *ppListe;
  tmp=(*tmpv).Naechstes;

  while( ! ((*tmp).Naechstes==NULL) )
  {
        if ( strcmp(tmp->Datenteil,Wert) == 0 )
        {
                tmpv->Naechstes=tmp->Naechstes;
                free(tmp);
                        goto ENDE;
        }

        tmpv=tmp;
        tmp=tmpv->Naechstes;
  }

  if ( strcmp(tmp->Datenteil,Wert) == 0 )
  {
        tmpv->Naechstes=NULL;
        free(tmp);
                goto ENDE;
  }

        ENDE:
                ;
}
```

```
/*Liste ausgeben*/

void ListeAnzeigen(ListenElement *pListe)
{
  if ( ! (pListe == NULL))
  {
        printf("\n\nDie auszugebende Liste lautet:\n\n");

        while ((*pListe).Naechstes != NULL )
        {
                printf("%s\n", pListe->Datenteil);
                pListe = pListe->Naechstes;
        }
        printf("%s\n", pListe->Datenteil);
  }
}


/*Liste löschen und Speicher freigeben*/

void ListeLoeschen(ListenElement **ppListe)
{
  if (*ppListe == NULL)
  {
        printf("\n\nDie Liste ist leer, loeschen nicht notwendig.\nSpeicher steht zur
Verfuegung!\n\n");
        goto END;
  }

  else if (*ppListe != NULL);
  {
  int i=0;
  int j=0;
  int k=0;

  ListenElement *tmp = NULL;

  if(*ppListe==NULL)
        ;

  printf ("\n\nDie Liste wird geloescht. Der Speicher wird freigegeben.\n\n");
  tmp = *ppListe;

  for(k=1;;++k)

  if (tmp->Naechstes == NULL)
        break;

  else
        tmp=tmp->Naechstes;

  for(i=1;i<=k;++i)
  {
        tmp=*ppListe;
        for(j=1;j<=k-i;++j)
                tmp=tmp->Naechstes;
        free(tmp);
  }

  *ppListe = NULL;

}
  END:
        ;
}



/*Hauptprogramm*/

int main(void)
```

```
{
   ListenElement *ListenAnfang = NULL;             /* Listenanfang */
   STRING EingegebeneDaten;                        /* Puffer fuer Dateneingabe */
   int cmp;
   int res;                                        /* zum Sichern von Funktionswerten */

   puts("\n\nGeben Sie anzufuegende Werte ein (ENDE = #)");
   do
   {
      fflush(stdin);
      scanf(FMTSTR, EingegebeneDaten);
      if ((cmp = strcmp(EingegebeneDaten, "#")) != 0)
      {
         if ((res = Anfuegen(&ListenAnfang, EingegebeneDaten)) == HEAPERR)
            puts("\n\n!!!Anfuegen: Kein dynamischer Speicherplatz!!!\n\n");
      }
   }

   while (cmp != 0 && res != HEAPERR);

   puts("\n\nGeben Sie einzufuegende Werte ein (ENDE = #)");
   do
   {
      fflush(stdin);
      scanf(FMTSTR, EingegebeneDaten);
      if ((cmp = strcmp(EingegebeneDaten, "#")) != 0)
      {
         if ((res = Einfuegen(&ListenAnfang, EingegebeneDaten)) == HEAPERR)
            puts("\n\n!!!Einfuegen: Kein dynamischer Speicherplatz!!!\n\n");
      }
   }

   while (cmp != 0 && res != HEAPERR);

   puts("\n\nGeben Sie zu loeschende Werte ein (ENDE = #)");
   do
   {
      fflush(stdin);
      scanf(FMTSTR, EingegebeneDaten);
      if ((cmp = strcmp(EingegebeneDaten, "#")) != 0)
      {
         Loeschen(&ListenAnfang, EingegebeneDaten);
      }
   } while (cmp != 0);

   ListeAnzeigen(ListenAnfang);
   ListeLoeschen(&ListenAnfang);

   return 0;
}
```

Teilaufgabe b:

/*evliste.h*/

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

# define MAXSTRLEN  5
# define FMTSTR     "%5s"
# define OK         0
# define HEAPERR   (-1)

typedef char STRING [MAXSTRLEN + 1];

typedef struct ListenElement
{
   STRING Datenteil;
   struct ListenElement *Naechstes;
}
```

```
ListenElement;


/*aufgabe2b.c*/
# include "evliste.h"
# include "evliste.c"

/*Hauptprogramm*/

int main(void)
{
   ListenElement *ListenAnfang = NULL;             /* Listenanfang */
   STRING EingegebeneDaten;                        /* Puffer fuer Dateneingabe */
   int cmp;
   int res;                                        /* zum Sichern von Funktionswerten */

   puts("\n\nGeben Sie anzufuegende Werte ein (ENDE = #)");
   do
   {
      fflush(stdin);
      scanf(FMTSTR, EingegebeneDaten);
      if ((cmp = strcmp(EingegebeneDaten, "#")) != 0)
      {
         if ((res = Anfuegen(&ListenAnfang, EingegebeneDaten)) == HEAPERR)
            puts("\n\n!!!Anfuegen: Kein dynamischer Speicherplatz!!!\n\n");
      }
   }

   while (cmp != 0 && res != HEAPERR);

   puts("\n\nGeben Sie einzufuegende Werte ein (ENDE = #)");
   do
   {
      fflush(stdin);
      scanf(FMTSTR, EingegebeneDaten);
      if ((cmp = strcmp(EingegebeneDaten, "#")) != 0)
      {
         if ((res = Einfuegen(&ListenAnfang, EingegebeneDaten)) == HEAPERR)
            puts("\n\n!!!Einfuegen: Kein dynamischer Speicherplatz!!!\n\n");
      }
   }

   while (cmp != 0 && res != HEAPERR);

   puts("\n\nGeben Sie zu loeschende Werte ein (ENDE = #)");
   do
   {
      fflush(stdin);
      scanf(FMTSTR, EingegebeneDaten);
      if ((cmp = strcmp(EingegebeneDaten, "#")) != 0)
      {
         Loeschen(&ListenAnfang, EingegebeneDaten);
      }
   } while (cmp != 0);

   ListeAnzeigen(ListenAnfang);
   ListeLoeschen(&ListenAnfang);

   return 0;
}


/*evliste.c*/
/*Listenelemente am Ende der Liste anfügen*/

int Anfuegen(ListenElement **ppListe, STRING Wert)
{
  int i;
  ListenElement *tmp;
  ListenElement *NEW;
```

```
  NEW = (ListenElement*) malloc(sizeof(ListenElement));

  if (NEW == NULL)
         return HEAPERR;

  if(*ppListe == NULL)
         {
           *ppListe = NEW;
           (**ppListe).Naechstes = NULL;
         }

  tmp = *ppListe;

  while ((*tmp).Naechstes != NULL )
         tmp = (*tmp).Naechstes;

  for (i=0; i<=MAXSTRLEN; ++i)

  *((*NEW).Datenteil+i) = *(Wert+i);
  (*NEW).Naechstes = NULL;

  if( ! (tmp==NEW))
         (*tmp).Naechstes=NEW;

return OK;
}


/*Listemelemente am Anfang der Liste einfügen*/

int Einfuegen(ListenElement **ppListe, STRING Wert)
{
  int i;
  ListenElement *NEW;

  NEW = (ListenElement*) malloc(sizeof(ListenElement));

  if (NEW == NULL)
         return HEAPERR;

  for (i=0; i<=MAXSTRLEN; ++i)
         *(NEW->Datenteil+i) = *(Wert+i);
         NEW->Naechstes = (*ppListe);

         *ppListe=&(*NEW);

         return OK;
}


/*Lisenelemente löschen*/

void Loeschen(ListenElement **ppListe, STRING Wert)
{
  ListenElement *tmpv = NULL;
  ListenElement *tmp = NULL;

  if (*ppListe == NULL)
         goto ENDE;

  if (strcmp((**ppListe).Datenteil,Wert) == 0)
  {
         if ((**ppListe).Naechstes == NULL)
         {
                  free(*ppListe);
                  *ppListe=NULL;
                           goto ENDE;
         }

  else
  {
```

```
        tmp=*ppListe;
        *ppListe=&(*(**ppListe).Naechstes);
        free(tmp);
                goto ENDE;
  }

  }

  if ((**ppListe).Naechstes==NULL)
        goto ENDE;

  tmpv = *ppListe;
  tmp=(*tmpv).Naechstes;

  while( ! ((*tmp).Naechstes==NULL) )
  {
        if ( strcmp(tmp->Datenteil,Wert) == 0 )
        {
                tmpv->Naechstes=tmp->Naechstes;
                free(tmp);
                        goto ENDE;
        }

        tmpv=tmp;
        tmp=tmpv->Naechstes;
  }

  if ( strcmp(tmp->Datenteil,Wert) == 0 )
  {
        tmpv->Naechstes=NULL;
        free(tmp);
                goto ENDE;
  }

        ENDE:
                ;
}


/*Liste ausgeben*/

void ListeAnzeigen(ListenElement *pListe)
{
  if ( ! (pListe == NULL))
  {
        printf("\n\nDie auszugebende Liste lautet:\n\n");

        while ((*pListe).Naechstes != NULL )
        {
                printf("%s\n", pListe->Datenteil);
                pListe = pListe->Naechstes;
        }
        printf("%s\n", pListe->Datenteil);
  }
}


/*Liste löschen und Speicher freigeben*/

void ListeLoeschen(ListenElement **ppListe)
{
  if (*ppListe == NULL)
  {
        printf("\n\nDie Liste ist leer, loeschen nicht notwendig.\nSpeicher steht zur
Verfuegung!\n\n");
        goto END;
  }

  else if (*ppListe != NULL);
  {
  int i=0;
```

```
  int j=0;
  int k=0;

  ListenElement *tmp = NULL;

  if(*ppListe==NULL)
        ;

  printf ("\n\nDie Liste wird geloescht. Der Speicher wird freigegeben.\n\n");
  tmp = *ppListe;

  for(k=1;;++k)

  if (tmp->Naechstes == NULL)
        break;

  else
        tmp=tmp->Naechstes;

  for(i=1;i<=k;++i)
  {
        tmp=*ppListe;
        for(j=1;j<=k-i;++j)
                tmp=tmp->Naechstes;
        free(tmp);
  }

  *ppListe = NULL;

}
  END:
        ;
}
```

Teilaufgabe c:

```
/*evlister.h*/
# include <stdio.h>
# include <string.h>
# include <stdlib.h>

# define MAXSTRLEN  5
# define FMTSTR     "%5s"
# define OK         0
# define HEAPERR   (-1)

typedef char STRING [MAXSTRLEN + 1];

typedef struct ListenElement
{
   STRING Datenteil;
   struct ListenElement *Naechstes;
}
ListenElement;


/*aufgabe2c.c*/
# include "evlister.h"
# include "evlister.c"

/*Hauptprogramm*/

int main(void)
{
   ListenElement *ListenAnfang = NULL;             /* Listenanfang */
   STRING EingegebeneDaten;                        /* Puffer fuer Dateneingabe */
   int cmp;
   int res;                                        /* zum Sichern von Funktionswerten */
   int Lauf=0;
```

```
   puts("\n\nGeben Sie anzufuegende Werte ein (ENDE = #)");
   do
   {
      fflush(stdin);
      scanf(FMTSTR, EingegebeneDaten);
      if ((cmp = strcmp(EingegebeneDaten, "#")) != 0)
      {
         if ((res = Anfuegen(&ListenAnfang, EingegebeneDaten)) == HEAPERR)
            puts("\n\n!!!Anfuegen: Kein dynamischer Speicherplatz!!!\n\n");
      }
   }

   while (cmp != 0 && res != HEAPERR);

   puts("\n\nGeben Sie einzufuegende Werte ein (ENDE = #)");
   do
   {
      fflush(stdin);
      scanf(FMTSTR, EingegebeneDaten);
      if ((cmp = strcmp(EingegebeneDaten, "#")) != 0)
      {
         if ((res = Einfuegen(&ListenAnfang, EingegebeneDaten)) == HEAPERR)
            puts("\n\n!!!Einfuegen: Kein dynamischer Speicherplatz!!!\n\n");
      }
   }

   while (cmp != 0 && res != HEAPERR);

   puts("\n\nGeben Sie zu loeschende Werte ein (ENDE = #)");
   do
   {
      fflush(stdin);
      scanf(FMTSTR, EingegebeneDaten);
      if ((cmp = strcmp(EingegebeneDaten, "#")) != 0)
      {
         Loeschen(&ListenAnfang, EingegebeneDaten);
      }
   } while (cmp != 0);

   printf("\n\nDie auszugebende Liste lautet:\n\n");
   ListeAnzeigen(ListenAnfang);

    printf("\n\nDie Liste wird geloescht. Der Speicher steht anschliessend wieder zur
Verfuegung...\n");
   ListeLoeschen(&ListenAnfang);
   printf("\nSpeicher steht zur Verfuegung!\n\n");

   return 0;
}
```

/*evlister.c*/

```
/*Listenelemente am Ende der Liste anfügen*/

int Anfuegen(ListenElement **ppListe, STRING Wert)
{
        int i;
        int erg;
        ListenElement *NeuesElement;

        if((*ppListe) == NULL)
        {
                NeuesElement = (ListenElement *) malloc(sizeof(ListenElement));

                if (NeuesElement==NULL)
                        erg=HEAPERR;
                else
                {
                        (*ppListe)=NeuesElement;
                        for (i=0; i <= strlen(Wert); ++i)
                                *(NeuesElement->Datenteil+i) = *(Wert+i);
```

```
                        NeuesElement->Naechstes = NULL;
                        erg=OK;
                }

        }
        else
                erg=Anfuegen(&(**ppListe).Naechstes, Wert);

        return erg;
}


/*Listemelemente am Anfang der Liste einfügen*/

int Einfuegen(ListenElement **ppListe, STRING Wert)
{
  int i;
  ListenElement *NEW;

  NEW = (ListenElement*) malloc(sizeof(ListenElement));

  if (NEW == NULL)
        return HEAPERR;

  for (i=0; i<=MAXSTRLEN; ++i)
        *(NEW->Datenteil+i) = *(Wert+i);
        NEW->Naechstes = (*ppListe);

        *ppListe=&(*NEW);

        return OK;
}


/*Lisenelemente löschen*/

void Loeschen(ListenElement **ppListe, STRING Wert)
{
  ListenElement *tmp=NULL;

        if(*ppListe!=NULL)
                if(strcmp((**ppListe).Datenteil, Wert) == 0)
                {
                        tmp=*ppListe;
                        *ppListe=(**ppListe).Naechstes;
                        free(tmp);
                }
                else
                        Loeschen(&(**ppListe).Naechstes,Wert);

}


/*Liste ausgeben*/

void ListeAnzeigen(ListenElement *pListe)
{

        if(pListe != NULL)
        {
                printf("%s\n", pListe->Datenteil);
                ListeAnzeigen(pListe->Naechstes);
        }

}


/*Liste löschen und Speicher freigeben*/

void ListeLoeschen(ListenElement **ppListe)
{
```

```
        if (*ppListe != NULL)
        {
                ListeLoeschen(&(**ppListe).Naechstes);
                free(*ppListe);
                *ppListe=NULL;
        }


}
```

Aufgabe 3:

/*sortbaum.h*/

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <ctype.h>
#include <conio.h>

#define MAXSTRLEN  20           /* muss mit FMTSTR    uebereinstimmen */
#define FMTSTR     "%20s"       /* muss mit MAXSTRLEN uebereinstimmen */

#define OK         0
#define HEAPERR   (-1)
#define LEFT 0
#define RIGHT 1
#define MIDDLE 2


typedef char tSTRING20[MAXSTRLEN + 1];
typedef struct BinBaum
{
        tSTRING20 Word;
        int Anzahl;
        struct BinBaum *Linker;
        struct BinBaum *Rechter;
} Baum;

Baum *New(void);
char *Konvertieren(tSTRING20 str);
int Sortieren(tSTRING20 tnWord, tSTRING20 Wert);
int Eintragen(Baum **pWurzel, tSTRING20 Wert);
void BaumAusgeben(Baum *pWurzel);
void BaumLoeschen(Baum **pWurzel);
int KnotenAnzahl (Baum *pWurzel);
```

/*main.c*/

```
#include "sortbaum.h"

int main(void)
{
        Baum *Wurzel = NULL;                    /* Baumanfang */
        tSTRING20 EingegebeneDaten;             /* Puffer fuer Dateneingabe */
        int cmp, res;                                   /*     zum     Sichern     von
Funktionswerten */
        int a=0;
        int b=0;
        int ka=0;

        while (a != 'b')
        {

        printf("\n\nDruecken Sie 's' zum starten und 'b' zum beenden\n");

        a = getch ();

        switch (a)
        {
        case 's':
                {

                        puts("\n\nGeben Sie die zu sortierenden Werte ein (ENDE = #)");
                        do
                        {
                                fflush(stdin);
                                scanf(FMTSTR, EingegebeneDaten);
                                if ((cmp = strcmp(EingegebeneDaten, "#")) != 0)
                                {
                                        if      ((res       =       Eintragen(&Wurzel,
```

```
EingegebeneDaten)) == HEAPERR)
                                                    puts("Anfuegen:    Kein    dynamischer
Speicherplatz!");
                                }
                        } while (cmp != 0 && res != HEAPERR);

                        if(Wurzel != NULL)
                        {
                                printf("\n\nDer auszugebende Baum lautet:\n\n");
                                printf("Anzahl:\t\tWort:\n\n");
                                BaumAusgeben(Wurzel);
                                printf("\n\nEs   wurden   %d   Knoten   benutzt\n\n\n",
KnotenAnzahl (Wurzel));
                                BaumLoeschen(&Wurzel);
                        }
                        else
                        {       printf("\n\nDer Baum ist leer\n\n\n");
                        BaumAusgeben(Wurzel);
                        }
                        break;
                }
        case 'b':
                {
                        printf("\n\nGoodbye!\n\n");
                        return 0;
                }

        default:
                {
                        printf("\n\nSie haben weder 's' noch 'b' gedrueckt...\n");
                        //return 0;
                }
        }
        }


        return 0;
}
```

/*sortbaum.c*/

```
#include "sortbaum.h"


Baum *New(void)
{
        Baum *BinBaumNeu=NULL;

        BinBaumNeu=(Baum *)malloc(sizeof(Baum));

        if(!(BinBaumNeu==NULL))
        {
                *(*BinBaumNeu).Word='\0';
                (*BinBaumNeu).Anzahl=1;
                (*BinBaumNeu).Linker=(*BinBaumNeu).Rechter=NULL;
        }

        return BinBaumNeu;
}
char *Konvertieren(tSTRING20 str)
{
        unsigned int i,j;
        char *s=NULL;

        s=(char *)malloc(2*MAXSTRLEN+1);

        j=0;
        for(i=0;i<strlen(str);++i)
        {

                switch(str[i])
```

```
                {
                case '':
                        *(s+(j++))=(char)'a';
                        *(s+(j++))=(char)'e';
                        break;

                case '':
                        *(s+(j++))=(char)'o';
                        *(s+(j++))=(char)'e';
                        break;

                case '':
                        *(s+(j++))=(char)'u';
                        *(s+(j++))=(char)'e';
                        break;

                case '':
                        *(s+(j++))=(char)'a';
                        *(s+(j++))=(char)'e';
                        break;

                case '':
                        *(s+(j++))=(char)'o';
                        *(s+(j++))=(char)'e';
                        break;

                case '':
                        *(s+(j++))=(char)'u';
                        *(s+(j++))=(char)'e';
                        break;

                case 'á':
                        *(s+(j++))=(char)'s';
                        *(s+(j++))=(char)'s';
                        break;


                default:
                        if(isupper((int)str[i]))
                                *(s+(j++))=(char) tolower((int)str[i]);

                        else
                                *(s+(j++))=str[i];

                        break;

                }
        }
        *(s+j)='\0';
        return s;

}
int Sortieren(tSTRING20 tnWord, tSTRING20 Wert)
{
        int erg;
        int len1,len2;
        int i,a,b;

        char *str1=NULL;
        char *str2=NULL;

        str1=Konvertieren(tnWord);
        str2=Konvertieren(Wert);

        len1=strlen(str1);
        len2=strlen(str2);

        if( strcmp(str1,str2)==0 )
        {
                erg = MIDDLE;
                i=len1+len2;
```

```
        }
        else
        {
                i=0;
        }
        while(i < len1 && i < len2)
        {

                a=(int) str1[i];
                b=(int) str2[i];

                if( a < b )
                {
                        erg=RIGHT;
                        i=len1+len2;
                }
                if( a > b )
                {
                        erg=LEFT;
                        i=len1+len2;
                }
                ++i;
        }

        if( i < (len1 +2) && i < (len2+2) )
        {
                if(len1<len2)
                        erg=RIGHT;
                if(len1>len2)
                        erg=LEFT;
        }

        free(str1);
        free(str2);
        return erg;
}

int Eintragen(Baum **pWurzel, tSTRING20 Wert)
{
        unsigned int i;
        int FINISH;
        Baum *tmp;
        Baum *BinBaumNeu;

        if(*pWurzel==NULL)
        {
                *pWurzel=New();
                if(*pWurzel==NULL)
                        return HEAPERR;

                for(i=0;i<=strlen(Wert);++i)
                        *( (**pWurzel).Word+i ) = *( Wert+i );

        }
        else
        {
                tmp=*pWurzel;
                FINISH=1;
                while(FINISH)
                {
                        switch( Sortieren((*tmp).Word, Wert) )
                        {
                        case MIDDLE:
                                ++(*tmp).Anzahl;
                                FINISH=0;
                                break;

                        case LEFT:
                                if((*tmp).Linker==NULL)
                                {
                                        BinBaumNeu=New();
```

```
                                        (*tmp).Linker=BinBaumNeu;
                                        for(i=0;i<=strlen(Wert);++i)
                                                *(    (*BinBaumNeu).Word+i    )    =    *
( Wert+i );
                                        FINISH=0;
                                }

                                else
                                {
                                        tmp=(*tmp).Linker;

                                }

                                break;

                        case RIGHT:
                                if((*tmp).Rechter==NULL)
                                {
                                        BinBaumNeu=New();
                                        (*tmp).Rechter=BinBaumNeu;
                                        for(i=0;i<=strlen(Wert);++i)
                                                *( (*BinBaumNeu).Word+i )  = *( Wert+i
);
                                        FINISH=0;
                                }

                                else
                                {
                                        tmp=(*tmp).Rechter;
                                }

                                break;

                        }
                }
        }

        return 0;
}

void BaumAusgeben(Baum *pWurzel)
{
        if(pWurzel != NULL)
        {
                BaumAusgeben( (*pWurzel).Linker );
                printf("%2d\t\t%s\n", pWurzel->Anzahl, pWurzel->Word);
                BaumAusgeben( (*pWurzel).Rechter );
        }

}
void BaumLoeschen(Baum **pWurzel)
{
        if(*pWurzel != NULL)
        {
                BaumLoeschen( &(**pWurzel).Linker );
                BaumLoeschen( &(**pWurzel).Rechter );
                free(*pWurzel);
                *pWurzel=NULL;

        }

}

int KnotenAnzahl (Baum *pWurzel)
{
        if (pWurzel == NULL)
                return 0;
        else
                return  KnotenAnzahl  ((*pWurzel).Linker)  +  KnotenAnzahl  ((*pWurzel).
Rechter) + 1;
}
```