3.2 Doppelzeiger

Wie im letzten Kapitel erläutert, sind Doppelzeiger Zeiger, die auf einen weiteren Zeiger verweisen. Doppelzeiger werden unter anderem dann benötigt, wenn

- eine Matrix, die sowohl in der Zeilenzahl als auch in der Spaltenzahl dynamisch ist, gespeichert werden soll. Bei Verwendung von Doppelzeigern kann die Spaltenzahl zudem von Zeile zu Zeile sogar variieren (s. nächstes Beispiel)
- durch Zeiger sortiert wird (Datenbank/ Tabellenverwaltung). Die Speicherelemente/-ketten selbst werden hierbei nicht sortiert, sondern ihre Referenzen (die Zeiger, die auf die Elemente verweisen, Geschwindigkeitsvorteil!). Die Adressen dieser Zeiger müssen ebenfalls verwaltet werden; dies wird durch Doppelzeiger realisiert. Bei der Sortierung wird in der Regel nicht nur nach einer Relation sortiert, sondern nach mehreren (z.B. bei einer Dateiverwaltung nach Dateiname, Erstellungsdatum, Größe, Besitzer, etc.).

3.2.1 Zeiger-Felder

Ein Beispiel für Zeiger-Felder ist der index-basierte Zugriff auf Zeichenketten beliebiger Länge. Die Anzahl der Zeichenketten ist üblicherweise ebenfalls unbekannt. Über den Index wird zum einen der Zugriff auf die einzelnen Zeichenketten realisiert, zum anderen können die Zeichenketten hierüber auch sortiert werden, ohne die Zeichenketten selbst umkopieien zu müssen (konstanter Zeitaufwand).

Beispiel:

Es sollen Zeichenketten nicht vorbestimmter Länge eingegeben, über ein Zeiger-Feld abgespeichert und danach ausgegeben werden. Die Eingabe einer Zeichenkette werde jeweils durch Drücken der Return-Taste abgeschlossen.

Struktur des Programmes:

1. Initialisierung und Speicherreservierung für ein (char *)-Feld,
2. Für jedes benötigte Feldelement
    1. zeichenweise Speicherplatz anfordern,
    2. Zeichen einlesen und auf Ende der Eingabe (Return-Taste) überprüfen,
    3. Zeichenkette abschließen.
3. Zeichenketten auf die Standard-Ausgabe in umgekehrter Reihenfolge der Eingabe ausgeben. Allokierte Speicherelemente freigeben.

Programmversionen:

a. In einer ersten Version soll das Programm 4 Zeichenketten abspeichern,
b. In einer zweiten Programmversion soll die Anzahl der eingebbaren und abspeicherbaren Zeichenketten nicht vorbestimmt sein.

Eine Lösung für die erste Programmversion:

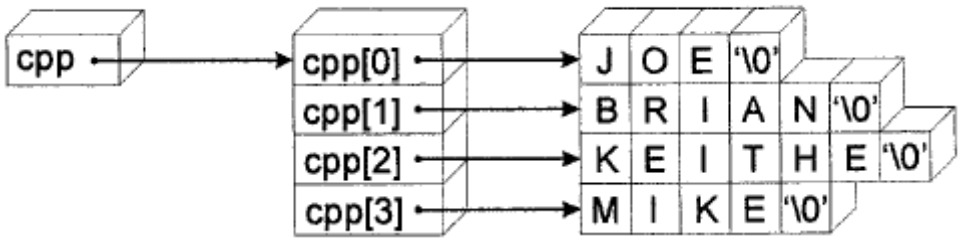
```
/*******************************************************
* ZgFeld.c         Informatik II:
* 05.11.2002
* H.-J. Meier, Copyright (c) Fachhochschule Schweinfurt
*******************************************************/
#include <stdio.h>
#define STRG_ANZAHL 4  /* Anzahl der vom Programm */
                       /* zu verwaltenden Strings */

int main (void){
   long lStrIdx, l;
   char **cppStrHdr;   /* Doppelzeiger auf Zeigerfeld */
   cppStrgHdr = (char **) calloc(STRG_ANZAHL, sizeof (char *));
   printf ("Bitte %2d Strings eingeben: \n", STRG_ÄNZAHL) ;
   for (i=0; i < STRG_ANZAHL; i++){
      lStrIdx = 0;
      cppStrgHdr[i] = (char *)malloc(sizeof(char));
      cppStrgHdr[i][0] =getchar();
      while (cppStrgHdr[i][lStrIdx] !='\n'){
         lStrIdx++;
         cppStrgHdr[i] = (char *) realloc(cppStrgHdr[i],
                                  (lStrIdx+1)*sizeof (char));
```

```
            cppStrgHdr[i][lStrIdx] = getchar();
         }
         cppStrgHdr[i][lStrIdx] = '\0';
      }

      /* In einem späteren Abschnitt (3.2.2.3) wird der Code
         an dieser Stelle um die Sortierung der Strings
         bzgl. ihrer lexikographischen Reihenfolge erweitert
         ...
      */

      for(i=(STRG_ANZAHL-1); i>=0; i—-){
         printf("%s\n", cppStrgHdr[i]);
         free(cppStrgHdr[i]);
      }
      free(cppStrgHdr};
}
```

Bei der Eingabe der vier Strings

JOE
BRIAN
KEITHE
MIKE

ergibt sich mit dem obigen Programm die nachfolgend dargestellte Speicherstruktur:



Frage und Übung:
Welche Veränderungen sind für die zweite Programmversion vorzunehmen? Die zweite Version des Programms ist in der 2. Übung zu implementieren.

3.2.2 Einfache Sortieralgorithmen

Beim Sortieren hängt die Wahl des Algorithmus stark von der Struktur der zu bearbeitenden Daten ab, so dass Sortiermethoden allgemein in zwei Kategorien eingeteilt werden:

- Sortieren von Feldern (internes Sortieren): Zugriff auf Daten kann schnell und willkürlich erfolgen (Abspeicherung der Felder im internen Speicher),
- Sortieren von sequentiellen Dateien (externes Sortieren): Zugriff auf Daten kann nur langsam und in sequentieller Art erfolgen (Daten sind im externen Speicher (Festplatten, etc.) hinterlegt).

Definition: **Sortieren**
Gegeben seinen n Elemente a[l], a[2], ..., a[n], so dass für eine gegebene Ordnungsfunktfion f gilt:

$$f(a[k1]) <= fl(a[k2]) <=...<= f(a[kn]).$$

Werden die Elemente in die Reihenfolge der Ordnungsfunktion umgeordnet, wird der Vorgang als Sortieren bezeichnet.

Die wichtigsten Anforderungen an Sortiermethoden für das Sortieren von Feldern sind:

- Wirtschaftliche (sparsame) Verwendung von Speicherplatz:
  - Bedeutet, dass die Sortierung in dem vorliegenden Feld zu erfolgen hat. Der Aufbau von mehreren Feldern, um Elemente von Feld a nach Feld b zu kopieren, ist nicht wirtschaftlich.
  - Zu sortierende Felder haben in der Regel Größenordnungen zwischen 1MB und 1(10) GB.
- Methode muß effizient sein:
  - Das heißt die Anzahl der Vergleiche und der Umsortierungen (Kopieren) soll möglichst gering sein.
  - Die Effizienz (Aufwand) wird mathematisch in Abhängigkeit der Feldgröße (Anzahl der Feldelemente) abgeschätzt und klassifiziert (O-Funktion), um die Effizienz von Sortieralgorithmen einordnen zu können.

- Einfache Sortieralgorithmen sind leichtverständlich in ihrer Struktur und leicht programmierbar, besitzen jedoch einen Aufwand von O(n * n), d.h. sie verhalten sich proportional zur quadratischen Anzahl der vorgegebenen Feldelemente.
- Effizientere Sortieralgorithmen besitzen einen Aufwand von O(n * log(n)), sind jedoch in ihrer Struktur komplizierter und erfordern mehr Programmaufwand.

Einfache Sortieralgorithmen mit Aufwand O(n * n), die im folgenden kurz diskutiert werden, sind:

- Sortieren durch Einfügen (insertion),
- Sortieren durch Auswählen (selection),
- Sortieren durch Austauschen (exchange).

3.2.2.1 Sortieren durch Einfügen

Diese Methode läßt sich am besten mit dem Sortieren der Spielkarten beim Kartenspielen vergleichen:

Die Elemente (Karten) werden in eine Zielsequenz [al] ... [a[i-l] und eine Quellensequenz a[i] ... a[n] aufgeteilt. Beginnend mit i = 2, wird bei jedem Schritt das Element a[i] der Quellensequenz herausgegriffen und an der entsprechenden Stelle in die Zielsequenz eingefügt. Damit ist die Zielsequenz stets sortiert. Dann wird i um eins erhöht.

Algorithmus:

```
for(i=1; i<=n; i++){
   x = a[i] ;
   "füge x am entsprechenden Platz in a[1] ... a[i] ein"
}
```

Wir betrachten ein Beispiel mit einem Feld von acht Elementen:

Anfangswerte:	44	55	12	42	94	18	06	67
I = 2	44	55	12	42	94	18	06	67
I = 3	12	44	55	42	94	18	06	67
I = 4	12	42	44	55	94	18	06	67
I = 5	12	42	44	55	94	18	06	67
I = 6	12	18	42	44	55	94	06	67
I = 7	06	12	18	42	44	55	94	67
I = 8	06	12	18	42	44	55	67	94

3.2.2.2 Sortieren durch Auswählen

Diese Methode beruht auf folgenden Schritten:

1. Auswahl des Elementes mit dem kleinsten Wert,
2. Austausch des Elementes gegen das erste Element a[l].
3. Wiederholung von 1. und 2. ohne a[1], d.h. a[2] übernimmt dessen Rolle ...

Algorithmus:

```
for(i=1; i<=n; i++){
   "weise x das kleinste Element von a[i] ... a[n] zu"
   "weise seinen Index k zu"
   "vertausche a[i] und a[k]"
}
```

Wir betrachten ein Beispiel mit einem Feld von acht Elementen:

Anfangswerte:	44	55	12	42	94	18	06	67
I = 2	06	55	12	42	94	18	44	67
I = 3	06	12	55	42	94	18	44	67
I = 4	06	12	18	42	94	55	44	67
I = 5	06	12	18	42	44	55	94	67
I = 6	06	12	18	42	44	55	94	67
I = 7	06	12	18	42	44	55	94	67
I = 8	06	12	18	42	44	55	67	94

3.2.2.3 Sortieren durch Austauschen

Der Kern (die Charakteristik) dieser Methode ist das Austauschen von Elementen.
Sie basiert auf dem fortgesetzten Vergleichen und Austauschen von Paaren *nebeneinander liegender* Elemente, bis alle Elemente sortiert sind.
Wie bei den ersten beiden Methoden wird das Feld mehrmals durchlaufen. Das kleinste Element der restlichen Menge wandert hierbei bei jedem Durchgang zum linken Ende des Feldes. Der Algorithmus wird auch BubbleSort genannt, da die einzelnen Elemente wie Blasen "aufsteigen".

Algorithmus:

```
for(i=1; i<=n; i++){
   for(j=n; j>i; j--){
      if   (a[j-1]>a[j])
      "vertausche a[j] und a[j-1]"
   }
}
```

Wir betrachten ein Beispiel mit einem Feld von acht Elementen:

Anfangswerte:	44	55	12	42	94	18	06	67
I = 2	06	44	55	12	42	94	18	67
I = 3	06	12	44	55	18	42	94	67
I = 4	06	12	18	44	55	42	67	94
I = 5	06	12	18	42	44	55	67	94
I = 6	06	12	18	42	44	55	67	94
I = 7	06	12	18	42	44	55	67	94
I = 8	06	12	18	42	44	55	67	94

Übung:
Die Zählvariable in der zweiten for-Schleife wird inkrementiert, entwickeln Sie die Struktur des Feldes für jeden Durchlauf.

Algorithmus:

```
for(i=1; i<=n; i++){
   for(j=i; j<n; j++){
     if (a[j] > a[j+1])
        "vertausche a[j] und a[j+1]"
   }
}
```

Anfangswerte:	44	55	12	42	94	18	06	67
I = 2								
I = 3								
I = 4								
I = 5								
I = 6								
I = 7								
I = 8								

Welches Ergebnis erhalten wir? Ist der Algorithmus zum Sortieren geeignet (Begründung!)?

Erweiterung des Programms `ZgFeld.c` um eine Sortierung:

```
for (i=0 ; i < STRG_ANZAHL ; i++){
   for (j=STRG_ANZAHL-1 ; j>i; j--){
      /*  cppstrgHdr[j-1]> cppstrgHdr[j] ? */
      if (strcmp(cppstrgHdr[j-1] , cppStrgHdr[j])>0){
         /* NUR die Zeiger werden vertauscht!! */
         char *cpStrg = cppStrgHdr[j-1];
         cppStrgHdr[j-1] = cppStrgHdr[j];
         cppStrgHdr[j] = cpStrg;
      }
   }
}
```

3.3 Doppelzeiger und zweidimensionale Felder

Doppelzeiger und zweidimensionale Felder sind in C sehr genau zu unterscheiden:

- Ein Doppelzeiger (char **cpp) ist eine Adressvariable vom Typ Zeiger auf Zeiger
- Eine zweidimensionales Feld (char cFeld[4][80]) ist eine Adresskonstante vom Typ Zeiger auf Feld

Diese Typen sind nicht zueinander kompatibel, d.h.

```
cpp = cFeld;
```

ist nicht zulässig. Zulässig ist dagegen:

```
cpp[i] = cFeld[i];
```

Wir betrachten ein Beispielprogramm

```
#include <stdio.h>
#include <string.h>
#define STRG_ANZAHL 5  /* # der einzugeb. Strings */
#define STRG_LAENGE 11  /* Max. Länge eines Strings */

void StrgInvrt1(char cStrgAry[][STRG_LAENGE]);
void StrgInvrt2(char **cppStrgH);

main(){
   char cStrgArray[STRG_ANZAHL][STRG_LAENGE];
   char **cppStrgHdr; /* Header für Zeigerfeld */
   long i;
   printf("%d Strings eingeben (max. %d Zeichen lang)\n",
                              STRG_ANZAHL,STRG_LAENGE-1);
   cppStrgHdr=(char **) calloc(STRG_ANZAHL,sizeof(char *));
   for(i=0; i<STRG_ANZAHL; i++)
      cppStrgHdr[i]=gets(cStrgArray[i]);

   printf("\nStrgInvrt1:\n");
   StrgInvrt1(cStrgArray) ;
```

```
   /* StrgInvrt2(cStrgArray) nicht zulässig! */
   for(i=0;i<STRG_ANZAHL; i++)
     printf("String %d: %s\n", i, cStrgArray[i]);

   printf("\nStrgInvrt2:\n");
   StrgInvrt2(cppStrgHdr);
   /*StrgInvrt1(cppStrgHdr) nicht zulässig!*/
   for (i=0; i<STRG_ANZAHL; i++)
      printf("String %d: %s\n",i, cStrgArray[i]);
}

void StrgInvrt1(char cStrgAry[][STRG_LAENGE]){
   long  i,j;
   for(i=0;i<STRG_ANZAHL;i++){
      j=0;
      while(cStrgAry[i][j]!='\0') {
         if ((cStrgAry[i][j]>='A') && (cStrgAry[i][j]<='Z'))
            cStrgAry[i][j]= cStrgAry[i][j]+'a'-'A';
         else if ((cStrgAry[i][j]>='a') && (cStrgAry[i][j]<='z'))
            cStrgAry[i][j]= cStrgAry[i][j]+'A'-'a';
         j++;
      }
   }
}

void StrgInvrt2(char **cppStrgH) {
   char  *cpStrg;
   long i;
   for(i=0;i<STRG_ANZAHL;i++){
      cpStrg = *(cppStrgH + i);
      while(*cpStrg!='\0'){
         if ((*cpStrg>='A') && (*cpStrg<='Z'))
            *cpStrg = *cpStrg + 'a'-'A';
         else *cpStrg = *cpStrg + 'A'-'a';
         cpStrg++;
      }
   }
}
```

Übung:

- Skizzieren Sie die erzeugte Speicherstruktur.
- Welche Inhalte besitzen die Speicherstellen der Adressen "cStrgArray+3" und "cppStrgHdr+2", wenn die Strings "Tom", "Frank", "Bob", "Mike" und "Test" eingegeben werden? Wie werden die Inhalte jeweils adressiert (z.B. für printf()-Ausgabe) ?
- Kompilieren Sie das Programm mit den als unzulässig gekennzeichneten Ausdrücken.