4.9 Datenstrukturen/In-Place-Substitution

4.9.1 Problem

Sie sollen eine Funktion schreiben, mit der Sie eine Zeichenkette auf durchgängige Großschreibung bringen (aus `"Hallo"` soll `"HALLO"` werden).

Mit der Anweisung `c=toupper(c)` können Sie das einzelne Zeichen `c` auf Großschreibung umwandeln (aus `'a'` wird `'A'`). Mit einer Zählschleife können Sie das für einen ganzen String machen:

```
1    for(i=0; x[i]!='\0'; ++i)
2        x[i]=toupper(x[i]);
```

Jetzt aber soll diese Änderung *in einer Funktion* eingeschlossen werden. Man muss also die Zeichenkette `x` als Parameter an die Funktion übergeben. Und zwar so, dass die Zeichenkette in der Funktion verändert wird und man nach dem Funktionsaufruf die veränderte Zeichenkette (also alles in Großbuchstaben) benutzen kann.

4.9.2 Funktion soll Variable beschreiben

Wie war das in C mit der Übergabe von Parametern? Man kann in C an eine Funktion immer nur die Kopie einer Variablen übergeben (*call by value*). Das Original wird dabei nicht angetastet.

Daher verwendete man in C einen Trick: Man übergibt die Adresse der Variablen an die Funktion und kann dann in der Funktion mit Hilfe des Inhalts-Operators auf das Original zugreifen:

```
1  void funk1(int a);    /* Kopie von a wird an Funktion uebergeben.    */
2                        /* a kann von funk1() nicht veraendert werden */
3  void funk2(int *pa);  /* Kopie der Adresse von a wird an Funktion    */
4                        /* uebergeben, a kann von funk2() aus          */
5                        /* veraendert werden.                          */
```

4.9.3 Funktion soll Array beschreiben

Wie sieht das nun bei Arrays aus? Muss man hier also so verfahren:

```
1  void make_upcase(char (*s)[]); /* s ist Adresse eines Arrays */
2  ...
3  make_upcase(&zeichenkette);    /* Aufruf in main() */
```

Zur Klärung kann man sich ein Beispiel ansehen (`in_place_subst.c`):

```
1  #include <stdio.h>
2  void vokal_nach_o(char *kette);
3
4  int main(void)
5  {
6      char zeichenkette[50]="Das ist das Haus vom Nikolaus.";
7
8      vokal_nach_o(zeichenkette);
9      printf("%s\n", zeichenkette);
10     return 0;
11 }
12 /***************************/
13 void vokal_nach_o(char *kette)
14 {
15     int i;
```

```
16      for(i=0; kette[i]!='\0'; ++i)
17      {
18          if(kette[i]=='a'||kette[i]=='e'||kette[i]=='i'||kette[i]=='u')
19              kette[i]='o';
20      }
21  }
```

Hier scheint es so, dass nicht die Adresse des Arrays übergeben wird, sondern das Array selbst. Aber dann kann das Programm doch gar nicht funktionieren, oder?

Terminal

```
schueler@debian964:~$ ./in_place_subst
Dos ost dos Hoos vom Nokoloos.
```

Warum funktioniert das Programm dennoch?

Die Lösung ist einfach: **Wenn man einen Arraynamen an eine Funktion übergibt, übergibt man bereits einen Zeiger als Kopie. Über diesen Zeiger kann die Funktion den Array-Inhalt verändern.** Daher genügt die bisher bekannte Konstruktion:

```
1  void make_upcase(char *buf);    /* Prototyp, Schreibweise 1 */
2  void make_upcase(char buf[]);   /* Prototyp, Schreibweise 2 */
3  ...
4  make_upcase(zeichenkette);    /* Aufruf in main() */
```

Hiermit können alle Daten des originalen Arrays von der Funktion aus bearbeitet werden. Der Fachbegriff lautet *In-Place-Substitution*.

4.9.4 Nachteile

Welche Nachteile hat die *In-Place-Substitution*?

a) Wenn man z.B. einen String an die Funktion übergibt und er dort verändert wird, ist das eine Änderung am Original. Der alte Zustand des Originals ist danach verloren.

b) Im Zusammenhang mit konstanten Arrays, z.B. literalen Stringkonstanten wie `"Hallo\n"`, ergibt sich eine Besonderheit. Sie wird im nächsten Kapitel behandelt.

4.9.5 Lösung

`upcase.c` zeigt die Lösung:

```
1  #include <stdio.h>
2  #include <ctype.h>
3  void upcase(char *kette)
4  {
5      int i;
6      for(i=0; kette[i]!='\0'; ++i)
7      {
8          kette[i]=toupper(kette[i]);
9      }
10 }
11 /*****************************/
12 int main(void)
13 {
14     char zeichenkette[50]="Das ist das Haus vom Nikolaus.";
15     upcase(zeichenkette);
16     printf("%s\n", zeichenkette);
```

```
17      return 0;
18  }
```