4.9.F Datenstrukturen/In-Place-Substitution – Ergänzungen und Bilder

4.9.F.1 Und wenn man trotzdem einen Zeiger übergibt?

Was passiert eigentlich, wenn man trotzdem einen Adressoperator vor den Namen des aktuellen Parameters setzt?

trotzdem0.c

```
1 #include <stdio.h>
2 int main(void)
3 {
4     char str[81]="";
5     printf("String␣eingeben:␣");
6     scanf("%80[^\n]", &str);
7     printf(">%s<\n", str);
8     return 0;
9 }
```

- Zeile 6: Ein `"&"`-Zeichen zu viel!

Ergebnis: Das Programm lässt sich trotzdem übersetzen und funktioniert. Aber warum funktioniert es? Der Grund liegt darin, dass die Variable `str` in `main` tatsächlich als Array vorhanden ist.

Und für ein Array `arr` gilt die Sonderregel, dass `&arr` dasselbe meint wie `&arr[0]`. Bei Arrays gilt nämlich eine Sonderregel mit drei Unterregeln: Ein Arrayname `arr` meint immer die Startadresse des Arrays `&arr[0]`, außer bei:

a) `&arr`, dies meint ebenfalls die Startadresse

b) `sizeof arr`, dies meint die Größe des Arrays, nicht die der Startadresse

c) `++arr`, `--arr`, `arr=xyz`, `arr.abc`, diese Operationen sind nicht erlaubt

Fazit: Man kann das `&`-Zeichen vor den aktuellen Array-Parameter setzen, es hat jedoch keinen Einfluss.

4.9.F.2 Und wenn man eine Funktion schreibt, die die Adresse der Adresse eines Arrays übernimmt?

Man kann sich überlegen, tatsächlich eine Adresse zu übergeben, die die Startadresse der Funktion enthält:

trotzdem1.c

```
1  #include <ctype.h>
2  #include <stdio.h>
3
4  void funk(char *(ps[]))
5  {
6      int i=0;
7      while((*ps)[i])
8      {
9          (*ps)[i]=toupper((*ps)[i]);
10         ++i;
11     }
12 }
13 int main(void)
14 {
```

```
15      char str[]="hallo";
16      char *p=str;
17
18  // funk(&str); // SEGV, weil &str[0] selbst uebergeben wird
19      funk(&p); // funktioniert, weil Adr. von p uebergeben wird
20                // und p die Adresse &str[0] enthaelt
21      printf("%s\n", str);
22      return 0;
23  }
```

- Zeile 19: Wir übergeben die Adresse von `p`, wobei `p` die Startadresse von `str` enthält.
- Zeile 4: Adresse eines Array von char, wird vom Compiler abgeändert zu Adresse einer Adresse von char.
- Zeile 18: siehe unten

Das Beispiel funktioniert tatsächlich. Man muss nur aufpassen, dass man wirklich die Adresse einer Adresse übergibt. Falls man Zeile 18 statt Zeile 19 verwendet, gibt es eine Speicherzugriffsfehler, denn `&str` meint dasselbe wie `str`, also in Wirklichkeit `&str[0]`, und das ist „nur“ die Adresse von char. Und hier dasselbe noch einmal in Zeigerschreibweise (Parameter, Zeile 4) und mit Zeigerarithmetik (Zeilen 6–10):

trotzdem2.c

```
1   #include <ctype.h>
2   #include <stdio.h>
3
4   void funk(char **ps)
5   {
6       while(**ps)
7       {
8           **ps=toupper(**ps);
9           ++*ps;
10      }
11  }
12  int main(void)
13  {
14      char str[]="hallo";
15      char *p=str;
16
17      funk(&str); // SEGV, weil &str[0] selbst uebergeben wird
18      funk(&p); // funktioniert, weil Adr. von p uebergeben wird
19                // und p die Adresse &str[0] enthaelt
20      printf("%s\n", str);
21      return 0;
22  }
```