4.10 Datenstrukturen/Konstante Zeichenketten

4.10.1 In-Place-Substitution bei String-Konstanten

Das Programm `in_place_subst.c` aus der In-Place-Substitution hat einen Haken: Wenn man an die Funktion `upcase()` eine Stringkonstante übergibt, stürzt das Programm ab, obwohl es *vorher* einwandfrei compiliert werden konnte:

```
1  #include <stdio.h>
2  void vokal_nach_o(char *kette);
3
4  int main(void)
5  {
6     char *zeichenkette="Das ist das Haus vom Nikolaus.";
7     vokal_nach_o(zeichenkette);
8     printf("%s\n", zeichenkette);
9     return 0;
10 }
11 /***************************/
12 void vokal_nach_o(char *kette)
13 {
14    int i;
15    for(i=0; kette[i]!='\0'; ++i)
16    {
17       if(kette[i]=='a'||kette[i]=='e'||kette[i]=='i'||kette[i]=='u')
18          kette[i]='o';
19    }
20 }
```

Man erhält eine Speicherverletzung, und das Betriebssystem stoppt den Prozess, aber warum? Der Grund ist: Die Funktion `upcase()` schreibt in die Stringkonstante `zeichenkette`. Nun könnte in einem Mikrocontroller-System die Konstante in einem Flash-Eprom liegen und gar nicht veränderbar sein. Auf dem PC-System liegt sie in einem schreibgeschützten Bereich, und genau deshalb hat der Schreibzugriff zum Programm-Abbruch geführt.

4.10.2 Gegenmaßnahme

Was kann man tun, damit das nicht wieder passiert? Man müsste verhindern, dass man die Adressen von Konstanten übergibt an Funktionen, die in die Konstanten schreiben wollen.

Dazu müsste man Konstanten unterscheiden können von Variablen. In C benutzt man für Konstanten oft literale (=wörtliche) Konstanten wie `"Hello, world!"` oder `3.14159`. Mit Hilfe des Präprozessors und seiner `#define`-Anweisung kann man sie durch symbolische Konstanten wie `HELLO_WORLD_BOTSCHAFT` oder `PI` verstecken. Diese Konstanten werden im Binärprogramm direkt eingesetzt.

Aber es gibt dazu noch die Möglichkeit, Konstanten wie Variablen zu deklarieren. Sie werden dann unter einer eigenen Speicherstelle abgelegt und sind unter ihrem Namen aufrufbar:

```
1     const char kennbuchstabe='c';
2     const int obergrenze=32767;
3     const double mehrwertsteuer=0.19;
```

Das Schlüsselwort `const` bewirkt also, dass statt einer Variablen eine Konstante vereinbart wird.

4.10.3 Zeiger auf konstanten Inhalt

Nun kann man Konstanten nicht nur für elementare Datentypen verwenden, sondern auch für Arrays, Records, Zeiger usw. Im Beispiel oben brauchten wir eine Konstante für einen Zeiger:

```
1   const char *zeichenkette = "Das ist das Haus vom Nikolaus.";
```

`zeichenkette` ist ein Zeiger auf eine Konstante des Typs `char` (und auf die Elemente dahinter, die vom gleichen Typ sind). Der Zeiger selbts ist also eine Variable. Man darf ihn erhöhen, auf das `'N'` von `Nikolaus` setzen oder sonstwohin:

```
1   ++zeichenkette;                    /* erlaubt */
2   zeichenkette = &zeichenkette[21]; /* erlaubt */
3   zeichenkette = NULL;               /* erlaubt */
4   zeichenkette = "Andere Adresse";   /* erlaubt */
```

Aber das, worauf er zeigt, das ist konstant und darf nicht verändert werden:

```
1   zeichenkette[0]='d';               /* verboten */
2   *zeichenkette='d';                 /* verboten */
```

4.10.4 Konstanter Zeiger

Es gibt noch eine weitere Möglichkeit, bei einer Vereinbarung eines Zeigers das Schlüsselwort `const` zu benutzen:

```
1   char arr[]="Das ist das Haus vom Nikolaus."
2   char * const zeichenkette = arr;
```

Hier ist `zeichenkette` ein konstanter Zeiger auf ein Element vom Typ `char`. Hier ist also der Zeiger konstant, d. h. er darf auf nichts anderes zeigen:

```
1   ++zeichenkette;                    /* verboten */
2   zeichenkette = &zeichenkette[21]; /* verboten */
3   zeichenkette = NULL;               /* verboten */
4   zeichenkette = "Andere Adresse";   /* verboten */
```

Stattdessen darf man den Inhalt verändern:

```
1   zeichenkette[0]='d';               /* erlaubt */
2   *zeichenkette='d';                 /* erlaubt */
```

4.10.5 Konstanter Zeiger auf konstanten Inhalt

Schließlich kann man sowohl Inhalt als auch Zeiger als konstant vereinbaren:

```
1   const char * const zeichenkette="Das ist das Haus vom Nikolaus.";
```

Dann darf man weder den Zeiger verändern noch den Inhalt, auf den er zeigt.

4.10.6 Übergabe eines Zeigers auf eine Konstante

Was passiert, wenn man eine Konstante an eine Funktion übergibt? Dazu wird das obige Beispiel durch das Schlüsselwort `const` in der Vereinbarung von Zeichenkette ergänzt:

```
1   const char *zeichenkette="Das ist das Haus vom Nikolaus.";
```

Nun wird das Programm erneut compiliert:

Terminal
```
heinz@debian964:~$ gcc in_place_subst_absturz2.c
in_place_subst_absturz2.c: In function 'main':
in_place_subst_absturz2.c:7:4: warning: passing argument 1 of
'vokal_nach_o' discards 'const' qualifier from pointer target
type [enabled by default]
in_place_subst_absturz2.c:2:6: note: expected 'char *' but
argument is of type 'const char *'
```

Woher kommt nun die Warnung?

- Übergeben wird als aktueller Parameter eine Konstante vom Typ `const char *`
- Er wird kopiert in einen formalen Parameter vom Typ `char *`

Man erhält also nur eine Warnung. Das Programm wird dennoch compiliert. Wenn man es aufruft, stürzt es wie bisher zuverlässig ab. Das heißt, man muss einfach auf die Warnung achten und ist dann vor diesem Problem sicher.

Warum wurde „nur“ eine Warnung ausgegeben, das Programm aber trotzdem compiliert? Das Schlüsselwort `const` wurde erst relativ spät in C eingeführt, und man wollte zu den unzähligen vorhandenen Bibliotheksfunktionen (die alle noch kein `const` in den Parameterlisten hatten) kompatibel bleiben.

4.10.7 Konstanten in Parameterlisten

Auch in den Parameterlisten von Funktionen ist es sinnvoll, Konstanten zu benutzen.

a) Bei einem formalen Parameter der Form `const char *x` (darf in der Parameterliste – nur dort – auch als `const char x[]` geschrieben werden) wird schon beim Compilieren verhindert, dass ein Schreibzugriff auf den erhaltenen Inhalt stattfinden kann. In-Place-Substitution ist dadurch nicht möglich.

```
1 void nurlesen_funk(const char *zeichenkette)
2 {
3     putchar(zeichenkette[0]); /* erlaubt  */
4     zeichenkette[0]='A';      /* verboten */
5 }
```

An einen solchen formalen Parameter darf man sowohl einen Zeiger auf konstanten Inhalt (wie `const char *x`) als auch einen Zeiger auf variablen Inhalt (wie `char *x`) ohne Probleme übergeben.

b) Bei einem formalen Parameter der Form `char *x` darf in den erhaltenen Inhalt geschrieben werden. In-Place-Substitution ist erlaubt.

```
1 void inplace_funk(char *zeichenkette)
2 {
3     putchar(zeichenkette[0]); /* erlaubt */
4     zeichenkette[0]='A';      /* erlaubt */
5 }
```

An einen solchen formalen Parameter darf man nur einen Zeiger auf variablen Inhalt (`char *x`) übergeben. Bei einem Zeiger auf konstanten Inhalt (`const char *x`) erhält man die oben gezeigte Warnmeldung.

Regel: Wenn die Funktion den Inhalt der Zeichenkette `x` *nicht* antastet, sollte diese in der Parameterliste mit `const char *x` deklariert werden. Nur dann kann der Benutzer der Funktion sicher sein, dass er ohne Probleme einen beliebigen String, also auch eine Stringkonstante, an die Funktion übergeben kann. In allen anderen Fällen nimmt man wie bisher `char *x`.