6.6 Extras/Deklarationen verstehen

6.6.1 Situation

Vereinbarungen wie die folgenden aus dem C-Standard-Buch sind schwer zu verstehen:

```
int main(int argc, char *argv[]){}
int atexit(void (*function)(void));
void (*signal(int signum, void (*handler)(int)))(int);
```

Die Erfinder von C hatten sich gewünscht, dass die Vereinbarung einer Variablen nach der gleichen Methodik verläuft wie später ihre Benutzung. Leider entstehen dadurch solche für Anfänger nur schwer verständliche Konstruktionen.

Es gibt Programmiersprachen, bei denen Vereinbarungen mit der gleichen Bedeutung viel einfacher sind (Pascal, Modula-2, Oberon, Ada, Simula). Leider haben viele der aktuellen Mode-Sprachen ausgerechnet die C-Schreibweise übernommen (Java, C++, C#, PHP). Also führt kein Weg am Entschlüsseln dieser Vereinbarungen vorbei.

6.6.2 Entschlüsselung von Hand

Zuerst soll `char *argv[]` zerlegt werden.

- Man beginnt mit der Variable: `argv` ist .... Die Variable ist meistens das einzige bisher unbekannte Wort in der Vereinbarung.
- Jetzt wird weitergegangen mit dem Element, das am stärksten an den Variablennamen gebunden ist. Am allerstärksten binden natürlich Klammern um ein Element. Ansonsten orientiert sich die Reihenfolge der Bindung an der Vorrangtabelle der Operatoren.[1] Am stärksten ist hier die eckige Klammer gebunden; der Satz geht also weiter: ... ein Array aus....
- Nun fährt man fort mit dem Element, das an `argv[]` am stärksten gebunden ist, also dem Stern; der Satz geht also weiter mit ... Zeigern auf...
- Nun bleibt nur noch der Datentyp übrig; der Satz wird beendet mit ... `char`. Die Vereinbarung lautet also: `argv` ist ein Array aus Zeigern auf `char`.

Ebenso kann man beim Parameter `void (*function)(void)` von `atexit()` vorgehen.

- `function` ist ...
- ... ein Zeiger auf ...
- ... eine Funktion (Parameter: `void`) mit Rückgabetyp ...
- ... `void`.

Zusammengefasst gelten also diese Grundregeln:

a) Beginnen mit dem Variablennamen

b) Von dort aus jeweils das am stärksten klammernde Zeichen oder Zeichenpaar anfügen. Klammern *um* ein Element werden vor allem Anderen abgearbeitet (Rangstufe 0).

c) Die Reihenfolge/Stärke der Klammerung orientiert sich an der Vorrangtabelle der Operatoren.

d) Am Ende sollte ein Datentyp übrigbleiben.

e) `*` bedeutet hier: *Zeiger auf*

[1] Hier geht es zwar nicht um Operatoren, sondern um den Zusammenbau von Variablentypen, aber die Tabelle gilt trotzdem.

f) `[]` bedeutet hier: *Array von*; in Klammern steht die Anzahl der Elemente des Arrays

g) `()` (rechts neben einem Element) bedeutet hier: *Funktion mit Rückgabetyp*; in Klammern steht die Parameterliste der Funktion

6.6.3 Das Programm `dcl`

Die Entschlüsselung der `signal`-Funktion bleibt trotzdem schwierig. Mit Hilfe des Programms `dcl`[2] kann man sich die Arbeit aber vereinfachen. Es übersetzt Deklarationen in für Menschen verstehbare Sätze:

Terminal
```
schueler@debian964:~$ gcc -o dcl getch.c gettokken.c dcl.c
schueler@debian964:~$ ./dcl
char *argv[]
argv:  array[] of pointer to char
```

Das Programmende erreicht man durch Strg-D am Zeilenanfang (bedeutet EOF) oder einfach durch Strg-C.

Weil `dcl` sehr einfach aufgebaut ist, kann es keine Variablen-Attribute wie `const` oder `static` vertragen; ebenso kann es nur leere Parameterlisten (also auch ohne `void` darin) verarbeiten.

Beispielsweise soll folgende Vereinbarung übersetzt werden:
`double (*getfunc(void))(double phi).`
Man leert also die Parameterlisten aus, es bleibt übrig: `double (*getfunc())()`

Terminal
```
schueler@debian964:~$ ./dcl
double (*getfunc())()
getfunc:  function returning pointer to function returning double
```

Übersetzt bedeutet das: `getfunc()` ist eine Funktion (Parameterliste: `void`) mit Rückgabetyp Funktion (Parameterliste: `double`) mit Rückgabetyp `double`. `getfunc()` gibt also einen Zeiger auf eine Funktion wie `sin()` oder `cos()` zurück.

6.6.4 Das Programm `undcl`

Das Programm `undcl` arbeitet genau umgekehrt. Man gibt eine Vereinbarung in für Menschen lesbarer Reihenfolge ein und erhält die C-Vereinbarung als Ergebnis. Leider kann `undcl` kein Englisch, so dass man Begriffe wie Zeiger oder Array über die entsprechenden Zeichenfolgen eingeben muss. Beispiel: `x` soll ein Array (9 Elemente) aus Arrays (8 Elemente) aus Zeigern auf `char` sein, also ein `[8]␣[9]␣*`.Zwischen Variablenname und dem Rest darf auch kein Doppelpunkt sein, anders als bei der Ausgabe von `dcl`.

Terminal
```
schueler@debian964:~$ gcc -o undcl getch.c gettokken.c undcl.c
schueler@debian964:~$ ./undcl
x [8] [9] * char
char (*x[8][9])
```

[2] Aus: Brian Kernighan, Dennis M. Ritchie : Programmieren in C. Mit dem C-Reference Manual in deutscher Sprache. Hanser, 2. Auflage, 1990.