C-Deklarationen lesen und schreiben

Version 1.2 — 6.7.2009

eMail: tb@ostc.de
Web: www.ostc.de

Hinweise auf inhaltliche Fehler, Schreibfehler und unklare Formulierungen sowie Ergänzungen, Kommentare, Wünsche und Fragen können Sie gerne an den Autor richten:

OSTC Open Source Training and Consulting GmbH
Thomas Birnthaler
eMail: tb@ostc.de
Web: www.ostc.de

C-Deklarationen

Folgende Deklarationen sind auf Anhieb verständlich.

```
int   i;
char* p;
```

i ist eine Variable vom Typ `int`, *p* ist ein Zeiger auf eine Variable vom Typ `char`. Die beiden folgenden Deklarationen sind ebenfalls noch verständlich.

```
int* ia[3];
int  (*ib)[3];
```

ia ist eine Feld von 3 Zeigern auf Variablen vom Typ `int`, *ib* ist ein Zeiger auf ein `int`-Feld der Länge 3. Deklarationen dieser Form rufen allerdings bereits Unsicherheit hervor, es fehlt das richtige Verständnis, man hat sie meist auswendig gelernt.

Bei noch komplexeren Deklarationen ist oft Raten der einzige Ausweg. Das ist schade, da die Theorie der Deklarationen eigentlich sehr einfach und überschaubar ist. Deklarationen beruhen auf der **Hierarchie der C-Operatoren**, d.h. es gilt für die Auswertungsreihenfolge.

Operator	Bedeutung	Priorität	Auswertung
`(...)`	Klammerung	höchste	Reihenfolge ändern
`()`	Funktion	mittlere	links nach rechts
`[]`	Array	mittlere	links nach rechts
`*`	Zeiger	niedrigste	rechts nach links

Lesen von C-Deklarationen

In der hierdurch definierten Reihenfolge sind die Teile einer Deklaration gemäß folgenden Regeln zu lesen.

1. **Attribute** sind `[]`, `()` und `*`, also Feld, Funktion und Zeiger.
2. Stellen Sie den Bezeichner der Deklaration fest. Sagen Sie „**Bezeichner ist**“, wobei Bezeichner der Name der Variablen ist.
3. Schauen Sie rechts vom Bezeichner nach `()` oder `[]` (können auch fehlen):
   – Sagen Sie „**ein Feld von**“, wenn Sie `[]` sehen.
   – Sagen Sie „**ein Feld von x**“, wenn Sie `[x]` sehen.
   – Sagen Sie „**ein x-mal-y Feld von**“, wenn Sie `[x][y]` sehen.
   – Sagen Sie „**ein x-mal-y mal . . . Feld von**“, wenn Sie `[x][y][...]` sehen.
   – Sagen Sie „**Funktionen mit Rückgabewert**“, wenn Sie `()` sehen und das zuletzt gefundene Attribut `[]` war.
   – Sagen Sie „**eine Funktion mit Rückgabewert**“, wenn Sie `()` sehen.
4. Schauen Sie auf die linke Seite des Bezeichners (nur Sterne sind interessant, können auch fehlen, Klammerung beachten):
   – Sagen Sie „**Zeiger auf**“ für jeden `*`, wenn das zuletzt gefundene Attribut `[]` war.
   – Sonst sagen Sie „**ein Zeiger auf**“, wenn Sie `*` sehen.
5. War der bisher gelesene Ausdruck eingeklammert, betrachten Sie ihn als abgearbeitet, ignorieren die Klammern und fahren bei 3. fort.

6. Es muß ein terminierendes Attribut der Form `char`, `short`, `int`, `long`, `float`, `double`, `struct`, `union`, `...` oder ein selbstdefinierter Typ, evtl. versehen mit einem der Modifizierer `signed`, `unsigned`, `static`, `auto`, `register`, `extern` übrigbleiben.
   – Sagen Sie „**eine Struktur vom Typ y**“ für `struct y`.
   – Sagen Sie „**eine Union vom Typ y**“ für `union y`.
   – Sagen Sie „**eine Aufzählung vom Typ y**“ für `enum y`.
   – Sagen Sie „**ein Typ**“ für den/die noch vorhandenen Typ(en)

Beispiele

```
int i;
    ↑i ist
↑ein Integer
```

```
int *i;
     ↑i ist
    ↑ein Zeiger auf
↑Integer
```

```
int *i[3];
     ↑i ist
       ↑ein Feld von 3
    ↑Zeigern auf
↑Integer
```

```
int (*i)[3];
       ↑i ist
      ↑ein Zeiger auf
          ↑ein Feld von 3
↑Integern
```

```
int *i();
     ↑i ist
       ↑eine Funktion mit Rückgabewert
    ↑Zeiger auf
↑Integer
```

```
int (*i)();
       ↑i ist
      ↑ein Zeiger auf
          ↑eine Funktion mit Rückgabewert
↑Integer
```

```
int **i;
       ↑i ist
      ↑ein Zeiger auf
    ↑einen Zeiger auf
↑Integer
```

```
int *(*i)();
        ↑i ist
       ↑ein Zeiger auf
           ↑eine Funktion mit Rückgabewert
     ↑Zeiger auf
↑Integer

int *(*i[])();
        ↑i ist
         ↑ein Feld von
       ↑Zeigern auf
              ↑Funktionen mit Rückgabewert
     ↑Zeiger auf
↑Integer

int *(*(*i)())[10];
          ↑i ist
         ↑ein Zeiger auf
             ↑eine Funktion mit Rückgabewert
       ↑Zeiger auf
                  ↑ein Feld von 10
     ↑Zeigern auf
↑Integer

int *(*i[5])[10];
        ↑i ist
         ↑ein Feld von 5
       ↑Zeigern auf
                ↑ein Feld von 10
     ↑Zeigern auf
↑Integer
```

Schreiben von C-Deklarationen

Analog zu den Regeln für das Lesen von Deklarationen lassen sich folgende Regeln für das Schreiben von Deklarationen angeben. Hat man eine umgangssprachliche Beschreibung der Deklaration, läßt sich diese damit in ihre syntaktische Form umsetzen.

1. Schreiben Sie „**Bezeichner**“, wobei Bezeichner der Name der Variablen ist.
2. Merken Sie sich immer, ob das letzte Attribut, das Sie geschrieben haben, ein „`*`“ war.
3. Schreiben Sie „`*`“ zur Linken dessen, was Sie bisher geschrieben haben, solange Sie in der Beschreibung **Zeiger auf** sehen.
4. Klammern Sie das bisher geschriebene, wenn Sie zuletzt einen „`*`“ geschrieben haben.
5. Schreiben Sie „`[x]`“ zur Rechten dessen, was Sie bisher geschrieben haben, wenn Sie **Feld von x** sehen,
   – schreiben Sie „`[x][y]`“ rechts, wenn Sie **ein x-mal-y Feld von** sehen,

– schreiben Sie „`[x][y][...]`“ rechts, wenn Sie **ein x-mal-y mal ... Feld von** sehen,
– schreiben Sie „`[]`“ rechts, wenn Sie **Feld von** sehen,
– schreiben Sie „`()`“ rechts, wenn Sie **Funktion mit Rückgabewert** sehen.

6. Machen Sie bei 3. weiter, solange die Beschreibung noch nicht abgearbeitet ist.
7. Schreiben Sie „`Typ`“ links von allem und „`;`“ rechts von allem.

Beispiele

```
i ist → ein Integer
↓     ↓
i     int i;
```

```
i ist → ein Zeiger auf → ein Integer
↓     ↓            ↓
i     *i           int *i;
```

```
i ist → ein Feld von 3 → Zeigern auf → Integer
↓     ↓            ↓          ↓
i     i[3]         *i[3]      int *i[3];
```

```
i ist → ein Zeiger auf → ein Feld von 3 → Integern
↓     ↓            ↓            ↓
i     *i           (*i)[3]      int (*i)[3];
```

```
i ist → eine Funktion (F) mit Rückgabewert (RW) → Zeiger auf → Integer
↓     ↓                                  ↓          ↓
i     i()                                *i()       int *i();
```

```
i ist → ein Zeiger auf → einen Zeiger auf → Integer
↓     ↓            ↓               ↓
i     *i           **i             int **i;
```

```
i ist → ein Zeiger auf → eine Funktion mit RW → Zeiger auf → Integer
↓     ↓            ↓                  ↓          ↓
i     *i           (*i)()             *(*i)()    int *(*i)();
```

```
i ist → ein Feld von → Zeigern auf → Funktionen mit RW → Zeiger auf → Integer
↓     ↓           ↓          ↓                ↓          ↓
i     i[]         *i[]       (*i[])()         *(*i[])()  int *(*i[])();
```

```
i ist → ein Zeiger auf → eine F. mit RW → Zeiger auf → ein Feld von 5 → Zeigern auf → Integer
↓     ↓            ↓             ↓          ↓             ↓          ↓
i     *i           (*i)()        *(*i)()    (*(*i)())[5]  *...       int *(*(*i)())[5];
```

```
i ist → ein Feld von 5 → Zeigern auf → Felder von 10 → Zeigern auf → Integer
↓     ↓           ↓          ↓             ↓          ↓
i     i[5]        *i[5]      (*i[5])[10]   *...       int *(*i[5])[10];
```

Einschränkungen

- Es gibt **keine Felder von Funktionen**, d.h die Deklaration `int a[5]()` ist ungültig. Es gibt aber Felder von Zeigern auf Funktionen, d.h. die Deklaration `int (*a[5])()` ist gültig.
- **Eine Funktion kann kein Feld zurückliefern**, d.h die Deklaration `int a()[]` ist ungültig. Eine Funktion kann aber einen Zeiger auf ein Feld zurückliefern, d.h. die Deklaration `int (*a())[]` ist gültig.
- **Eine Funktion kann keine andere Funktion als Ergebnis liefern**, d.h die Deklaration `int a()()` ist ungültig. Eine Funktion kann aber einen Zeiger auf eine andere Funktion zurückliefern, d.h. die Deklaration `int (*a())()` ist gültig.

Typdefinitionen und Casts

- Bei einer Typdefinition ist statt dem Variablennamen der **Typname** einzusetzen und vor den ganzen Ausdruck `typedef` zu schreiben. Beispiel

```
int* f();   /* Funktion f mit Rueckgabewert Zeiger auf int */

typedef int* FUNC_TYPE();   /* Analoge Typdefinition */
FUNC_TYPE f;                /* 1:1 obige Deklaration */
```

- Für einen **cast (Typumwandlung)** ist die Deklaration einer entsprechenden Variablen (unter Weglassen des Variablennamens und in Klammern gesetzt), vor die umzuwandelnde Variable zu setzen. Beispiel

```
(*(void(*)()) 0xFFF0)();
```

  Eine Funktion (ohne Rückgabewert und ohne Argumente), deren Startadresse an der Speicherstelle `0xFFF0` liegt, soll aufgerufen werden. Dazu ist der Integerwert `0xFFF0` in einen Zeiger auf eine Funktion ohne Rückgabewert und ohne Argumente umzuwandeln. Das erledigt der *cast*-Ausdruck `(void(*)())`. Zum Aufruf ist anschließend noch der Zeiger zu dereferenzieren (`*`) und der Funktionsaufruf auszulösen (`()`). Eine identische, aber übersichtlichere Formulierung des Aufrufs läßt sich durch Einführen eines eigenen Typs `FPTR` erreichen

```
typedef void (*FPTR)();   /* Typdefinition */
(*(FPTR) 0xFFF0)();       /* Aufruf */
```

- Analog sind in ANSI-C bei der Angabe von Argumenttypen in Funktionsdeklarationen für jedes einzelne Argument seine Typdeklaration (wahlweise mit oder ohne Variablenname) anzugeben. Beispiel

```
void (*signal(int, void(*)(int)))(int);
```

Die Funktion `signal` hat zwei Argumente, eines vom Typ `int` und das andere vom Typ Zeiger auf eine Funktion mit Argument `int` und Rückgabewert `void`. Sie erwartet als Argument einen Integerwert und gibt keinen Wert (`void`) zurück. Eine identische, aber übersichtlichere Deklaration läßt sich durch Einführen eines eigenen Typs `SIGNAL_HANDLER` erreichen.

```
typedef void (*SIGNAL_HANDLER)(int);
SIGNAL_HANDLER signal(int, SIGNAL_HANDLER);
```