Bedeutung und Korrektheit von C Programmen

Vorlesung vom 22.04.2008: Typen in C: Deklarationen

Christoph Lüth & Lutz Schröder

SS 08

Fahrplan Teil I

- Teil I: C
  - Einführung in den Standard
  - Typen in C: Deklarationen
  - Typen in C: Konversionen, Felder vs. Zeiger
- Teil II
- Teil III

Fahrplan heute

- Deklarationen lesen
- Deklarationen verstehen

Deklarationen in C

- Sprachphilosophie: Declaration resembles use
- Deklarationen:
  - *declarator* — was deklariert wird
  - *declaration* — wie es deklariert wird

Der *declarator*

Anzahl	Name	Syntax
Kein oder mehr	*pointer*	`*`
		type-qualifier `*`
Ein	*direct-declarator*	*identifier*
		identifier `[`*expression*`]`
		identifier `(`*parameter-type-list*`)`
Höchstens ein	*initializer*	`=` *expression*

Die *declaration*

Anzahl	Name	Syntax
Ein oder mehr	*type-specifier*	`void`, `char`, `short`, `int`, `long`, `double`, `float`, `signed`, `unsigned`, *struct-or-union-spec*, *enum-spec*, *typedef-name*
Beliebig	*storage-class*	`extern`, `static`, `register`, `auto`, `typedef`
Beliebig	*type-qualifier*	`const`, `volatile`
Genau ein	*declarator*	s.o.
Beliebig	*declarator-list*	`,` *declarator*
Genau ein		`;`

Restriktionen

Illegal:

- Funktion gibt Funktion zurück: `foo ()()`
- Funktion gibt Feld zurück: `foo()[]`
- Felder von Funktionen: `foo[]()`

Aber erlaubt:

- Funktion gibt Zeiger auf Funktion zurück: `int (* fun))();`
- Funktion gibt Zeiger auf Feld zurück: `int (*foo())[];`
- Felder von Zeigern auf Funktionen: `int (*foo[])();`

Strukturen

- Syntax: `struct` *identifier*$_{\mathrm{Opt}}$ { *struct-declaration** }
- Einzelne Felder (*struct-declaration*):
  - Beliebig viele *type-specifier* oder *type-qualifier*, dann
  - *declarator*, oder
  - *declarator*$_{\mathrm{Opt}}$ `:` *expression*
- Strukturen sind first-class objects
  - Können zugewiesen und übergeben werden.
- `union`: syntaktisch wie `struct` (andere Semantik!)

Präzendenzen für Deklarationen

A	Name zuerst (von links gelesen)
B	In abnehmender Rangfolge:
B.1	Klammern
B.2	Postfix-Operatoren: `()` für Funktion `[]` für Felder
B.3	Präfix-Operator: `*` für Zeiger-auf
C	*type-qualifier* beziehen sich auf *type-specifier* rechts daneben (wenn vorhanden), ansonsten auf den Zeiger `*` links davor

Typdefinitionen mit `typedef`

- `typedef` definiert Typsynonyme
- `typedef` ändert eine Deklaration von
  - `x` ist eine Variable vom Typ `T` zu
  - `x` ist ein Typsynonym für `T`
- Braucht nicht am Anfang zu stehen (aber empfohlen)
- Nützliche Tipps:
  - Kein `typedef` für `structs`
  - `typedef` für lesbarere Typen
  - `typedef` für Portabilität (e.g. `uint32_t` in `<stdint.h>`)

Namensräume in C

- Labels;
- *tags* für Strukturen, Aufzählungen, Unionen;
- Felder von Strukturen (je eines pro Struktur/Union);
- Alles andere: Funktionen, Variablen, Typen, ...
- Legal: `struct foo {int foo; } foo;`
  - Was ist `sizeof(foo);`?

Zusammenfassung

- Deklarationen in C:
  - *declarator*, *declaration*
  - Präzendenzregeln: Postfix vor Präfix, rechts nach links