Verifikation von C-Programmen

Vorlesung 2 vom 30.10.2014: Der C-Standard: Typkonversionen und Programmausführung

Christoph Lüth

Universität Bremen

Wintersemester 2014/15

Fahrplan heute

- Typkonversionen (Typwechsel ohne Anmeldung)
- Felder vs. Zeiger — Das C-Speichermodell
- Zeiger in C:
  - Dynamische Datenstrukturen
  - Arrays
  - Funktionen höherer Ordnung

Verschiedene Typen im Standard

- Object types, incomplete types, function types (§6.2.5)
  - Basic types: standard/extended signed/unsigned integer types, floating types (§6.2.5)
  - Derived types: structure, union, array, function types
- Qualified Type (§6.2.5)
  - `float const *` qualified (qualified pointer to type)
  - `const float *` not qualified (pointer to qualified type)
- Compatible Types (§6.2.7)
- Assignable Types (§6.5.16, §6.5.2.2)

Typkonversionen in C

- Integer promotion (§6.3.1.1):
  - `char`, `enum`, `unsigned char`, `short`, `unsigned short`, *bitfield*
- `float` promoted to `double`
- `T []` promoted to `T *`
- Usual arithmetic conversions (§6.3.1.8)
- Konversion von Funktionsargumenten:
  - Nur bei K&R-artigen Deklarationen, nicht bei Prototypen!
  - Moral: Stil bei Prototyp und Definition nicht mischen!

Zeiger und Felder

- Zeiger sind Adressen

```
int *x;
```

- Feld: reserviert Speicherbereich für `n` Objekte:

```
int y[100];
```

  - Index beginnt immer mit 0
  - Mehrdimensionale Felder ... möglich
- Aber: `x` $\not\sim$ `y`

Wann sind Felder Zeiger?

- Wann kann ein Array in ein Zeiger konvertiert werden?

Externe Deklaration: `extern char a[]`	Keine Konversion
Definition: `char a [10]`	Keine Konversion
Funktionsparameter: `f(char a[])`	Konversion möglich
In einem Ausdruck: `x= a[3]`	Konversion möglich

- Wenn Konversion möglich, dann durch Semantik erzwungen
- Tückisch: Externe Deklaration vs. Definition
- Größe: `sizeof`

Mehrdimensionale Felder

- Deklaration: `int foo[2][3][5];`
- Benutzung: `foo[i][j][k];`
- Stored in Row major order (Letzter Index variiert am schnellsten) (§6.5.2.1)
- Kompatible Typen:
  - `int (* p)[3][5] = foo;`
  - `int (*r)[5]= foo[1];`
  - `int *t = foo[1][2];`
  - `int u = foo[1][2][3];`

Mehrdimensionale Felder als Funktionsparameter

- Regel 3 gilt nicht rekursiv:
  - Array of Arrays ist Array of Pointers, nicht Pointer to Pointer
- Mögliches Matching:

Parameter	Argument
`char (*c)[10];`	`char c[8][10]; char (*c)[10];`
`char **c;`	`char *c[10]; char **c;`
`char c[][10];`	`char c[8][10]; char (*c)[10];`

- `f(int x[][]);` nicht erlaubt

Mehrdimensionale Felder als Funktionsparameter

- Regel 3 gilt nicht rekursiv:
  - Array of Arrays ist Array of Pointers, nicht Pointer to Pointer
- Mögliches Matching:

Parameter	Argument
`char (*c)[10];`	`char c[8][10]; char (*c)[10];`
`char **c;`	`char *c[10]; char **c;`
`char c[][10];`	`char c[8][10]; char (*c)[10];`

- `f(int x[][]);` nicht erlaubt (§6.7.5.2)
  - NB. Warum ist `int main(int argc, char *argv[])` erlaubt?

Programmausführung (§5.1.2.3)

- Standard definiert abstrakte Semantik
- Implementation darf optimieren
- Seiteneffekte:
  - Zugriff auf volatile Objekte;
  - Veränderung von Objekten;
  - Veränderung von Dateien;
  - Funktionsaufruf mit Seiteneffekten.
- Reihenfolge der Seiteneffekte nicht festgelegt!
- Sequenzpunkten (Annex C) sequentialisieren Seiteneffekte.

Semantik: Statements

- Full statement, Block §6.8
- Iteration §6.8.5

Semantik: Speichermodell

- Der Speicher besteht aus Objekten
- lvalue §6.3.2.1: *Expression with object type or incomplete type other than* `void`
- lvalues sind Referenzen, keine Adressen
- Werden ausgelesen, außer
  - als Operand von `&`, `++`,`--`, `sizeof`
  - als Linker Operand von `.` und Zuweisung
  - *lvalue (has) array tyoe*
- Woher kommen lvalues?
  - Deklarationen
  - Indirektion (`*`)
  - `malloc`
- Adressen:
  - Adressoperator (`&`)
  - Zeigerarithmetik (§6.5.6)

Ausdrücke (in Auszügen)

- Einfache Bezeichner: lvalue
  - Bezeichner von Array-Typ: Zeiger auf das erste Element des Feldes (§6.3.2.1)
- Felder: `6.5.2.1`
  - `a[i]` definiert als `*((a)+(i))`
  - Damit: `a[i]=*((a)+(i))=*((i)+(a))=i[a]`
- Zuweisung: `6.5.16`
  - Reihenfolge der Auswerung nicht spezifiziert!

Funktionsaufrufe §6.5.2.2

- Implizite Konversionen:
  - Nur wenn kein Prototyp
  - Integer Promotions, `float` to `double`
- Argumente werden ausgewertet, und den Parametern zugewiesen
  - Funktionsparameter sind wie lokale Variablen mit wechselnder Initialisierung
- Reihenfolge der Auswertung von Funktionausdruck und Argumenten nicht spezifiziert, aber Sequenzpunkt vor Aufruf.

Zusammenfassung

- Typkonversionen in C: meist klar, manchmal überraschend
- Auswertung durch eine *abstrakte Maschine* definiert
- Speichermodell:
  - Speicher besteht aus Objekten
  - Durch `char` addressiert (*byte*)
  - Referenzen auf Objekte: *lvalue*