

Numerische Datentypen

Simon Weidmann

Universität Hamburg

8. Mai 2014

Agenda

- 1 Generell: Typen
 - Elementares
 - Gleitkommatypen
 - Signed vs. Unsigned
 - Überlauf und Zahlenkreis
- 2 Typecasting
 - Casten
 - Literale
- 3 Operationen
- 4 Architekturabhängigkeiten
- 5 Zusammenfassung
- 6 Literatur

Elementares

Was ist ein Typ?

Jeder Datentyp besteht aus **zwei** Teilen:

Elementares

Was ist ein Typ?

Jeder Datentyp besteht aus **zwei** Teilen:

- Aus einem Wertebereich z.B. [0,255]

Elementares

Was ist ein Typ?

Jeder Datentyp besteht aus **zwei** Teilen:

- Aus einem Wertebereich z.B. $[0,255]$
- Aus einer Menge an Operationen z.B. „+“, „*“, ...

Elementares

Was ist ein Typ?

Jeder Datentyp besteht aus **zwei** Teilen:

- Aus einem Wertebereich z.B. [0,255]
- Aus einer Menge an Operationen z.B. „+“, „*“, ...
- Bekannte numerische Datentypen: char, short, int, long, float, double

chars

chars haben 8 Bit und verhalten sich wie andere ganzzahlige Typen, sie werden jedoch (auch) als Buchstaben benutzt.

Gleitkommatypen

Wie werden Kommazahlen auf dem Rechner dargestellt?

Vorzeichen * 1.*Mantisse* * 2^{Exponent}

Gleitkommatypen

Wie werden Kommazahlen auf dem Rechner dargestellt?

$$\text{Vorzeichen} * 1.\text{Mantisse} * 2^{\text{Exponent}}$$

Ist 0.1 dadurch ausdrückbar?

Gleitkommatypen

Wie werden Kommazahlen auf dem Rechner dargestellt?

$$\text{Vorzeichen} * 1.\text{Mantisse} * 2^{\text{Exponent}}$$

Ist 0.1 dadurch ausdrückbar?

Achtung!

Rundungsfehler \Rightarrow „float“ ist nicht sehr präzise!

Float Ungenauigkeit

```
int i = 0;
```

```
i + 0.1f; → i ≠ 0.8
```

```
float f = 30.05 * 100; → f ≠ 3005
```

IEEE 754

Es gibt (historisch gewachsen) verschiedene Standards für Gleitkommatypen.

Aktuell verwendet wird **IEEE 754**:

float: 8 Bit Exponent, 23 Bit Mantisse

→ $[1.175 * 10^{-38}, 3.403 * 10^{38}]$

double: 11 Bit Exponent, 52 Bit Mantisse

→ $[2.2251 * 10^{-308}, 1.798 * 10^{308}]$

Signed vs. Unsigned

Unterschiede im Wertebereich

Wertebereich Unsigned: $[0, X]$

Wertebereich Signed: $[-X, X-1]$

Beispiel: 8Bit - Typ:

Signed vs. Unsigned

Unterschiede im Wertebereich

Wertebereich Unsigned: $[0, X]$

Wertebereich Signed: $[-X, X-1]$

Beispiel: 8Bit - Typ:

- Unsigned $[0, 255]$
- Signed $[-128, 127]$

Überlauf

Was, wenn der Wertebereich überschritten wird?

Bei unsigned Typen: Überlauf!

Man zählt $MAX+1=0$

Überlauf

Was, wenn der Wertebereich überschritten wird?

Bei unsigned Typen: Überlauf!

Man zählt $MAX+1=0$

Beispiel

$([0,255]) \rightarrow 200 + 200 = 154 = 400 - 256$

Überlauf

Was, wenn der Wertebereich überschritten wird?

Bei unsigned Typen: Überlauf!

Man zählt $MAX+1=0$

Beispiel

$([0,255]) \rightarrow 200 + 200 = 154 = 400 - 256$

Bei signed Typen: UNDEFINED!

Es kann also jederlei Unsinn herauskommen!

Achtung Fehlerquelle

Also: Aufpassen mit Wertebereichen

Zahlenkreis

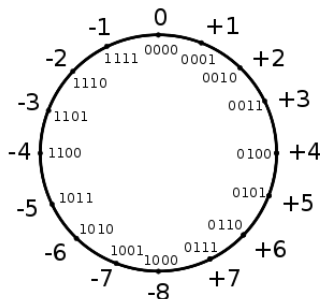


Abbildung:

<http://upload.wikimedia.org/wikipedia/commons/thumb/5/5d/4Bit-2Komplement.svg/300px-4Bit-2Komplement.svg.png>

Typecasting

Typen können umgewandelt werden:

Typecasting

Typen können umgewandelt werden:

implizit:

long + short = long

Typecasting

Typen können umgewandelt werden:

implizit:

long + short = long

explizit:

float f = 2.5;

double d = (double) f;

Literale

25 → Dezimal (normal)

0500 → Oktal

0xff3 → Hexadezimal

Literale

25 → Dezimal (normal)

0500 → Oktal

0xff3 → Hexadezimal

40 → int

40u → unsigned int

Literele

25 → Dezimal (normal)

0500 → Oktal

0xff3 → Hexadezimal

40 → int

40u → unsigned int

40l → long

40ul → unsigned long

Literale

25 → Dezimal (normal)

0500 → Oktal

0xff3 → Hexadezimal

40 → int

40u → unsigned int

40l → long

40ul → unsigned long

3.1d → double

3.1f → float

Operationen, Operatoren

Operationen, Operatoren

arithmetisch:

- „+“
- „-“

Operationen, Operatoren

arithmetisch:

- „+“
- „-“

bitweise:

- „&“
- „|“

Operationen, Operatoren

arithmetisch:

- „+“
- „-“

bitweise:

- „& “
- „| “

boolsch:

- „& & “
- „||“

Operationen, Operatoren

arithmetisch:

- „+“
- „-“

bitweise:

- „& “
- „| “

boolsch:

- „& & “
- „||“

Dies ist die Reihenfolge der Priortitäten!

Beispiele

Beispiele

Verschiedene Operatoren, verschiedene Ergebnisse:

- $31 + 12 = 43$

Beispiele

Verschiedene Operatoren, verschiedene Ergebnisse:

- $31 + 12 = 43$
- $31 \& 12 = 12$,
da $11111 \& 01100 = 01100$

Beispiele

Verschiedene Operatoren, verschiedene Ergebnisse:

- $31 + 12 = 43$
- $31 \& 12 = 12$,
da $11111 \& 01100 = 01100$
- $31 \&\& 12 = 1$

Beispiele

Verschiedene Operatoren, verschiedene Ergebnisse:

- $31 + 12 = 43$
- $31 \& 12 = 12$,
da $11111 \& 01100 = 01100$
- $31 \&\& 12 = 1$
- $31\&6 + 6 = 32\&12 = 12$

Beispiele

Verschiedene Operatoren, verschiedene Ergebnisse:

- $31 + 12 = 43$
- $31 \& 12 = 12$,
da $11111 \& 01100 = 01100$
- $31 \&\& 12 = 1$
- $31\&6 + 6 = 32\&12 = 12$

Präzedenzen

→ „C Language Operator Precedence Chart“ von Swanson Tec

verschiedene Architekturen

Gleiche Typen haben verschiedene Datengrößen

verschiedene Architekturen

Gleiche Typen haben verschiedene Datengrößen

Tipp vorm Programmieren

Mache dir bewusst, welches Format deine Typen haben, bevor du sie benutzt.

verschiedene Architekturen

Was tun, um nun trotzdem portabel zu programmieren?

verschiedene Architekturen

Was tun, um nun trotzdem portabel zu programmieren?

„portable redefined types“

Ganz einfach das Problem umgehen! → typedefs.h - Bibliothek

verschiedene Architekturen

Was tun, um nun trotzdem portabel zu programmieren?

„portable redefined types“

Ganz einfach das Problem umgehen! → typedefs.h - Bibliothek

Mit diesen viel tolleren Typen!

intmax_t , uintmax_t, uint64_t , int32_t, ...

Zusammenfassung

- 1 Typen bestehen aus Wertebereich und Operationen
- 2 Gleitkommatypen können ungenau sein.
- 3 Jeder Typ kann signed oder unsigned auftreten.

Zusammenfassung

- 1 Typen bestehen aus Wertebereich und Operationen
- 2 Gleitkommatypen können ungenau sein.
- 3 Jeder Typ kann signed oder unsigned auftreten.
- 4 Man muss auf eventuelle Überläufe aufpassen.
- 5 Typen werden (wenn möglich) implizit umgewandelt, können aber auch explizit gecastet werden.

Zusammenfassung

- 1 Typen bestehen aus Wertebereich und Operationen
- 2 Gleitkommatypen können ungenau sein.
- 3 Jeder Typ kann signed oder unsigned auftreten.
- 4 Man muss auf eventuelle Überläufe aufpassen.
- 5 Typen werden (wenn möglich) implizit umgewandelt, können aber auch explizit gecastet werden.
- 6 Je nach Architektur können Wertebereiche abweichen
- 7 ⇒ portable redefined types

Literatur I



Prof. Dr. Nikolaus Wulff

Programmieren in C

[http://www.lab4inf.fh-muenster.de/lab4inf/docs/Prog-in-C/
03-Operatoren_und_If-Else.pdf](http://www.lab4inf.fh-muenster.de/lab4inf/docs/Prog-in-C/03-Operatoren_und_If-Else.pdf)

Zuletzt: 29.04.2014



Wikimedia

4Bit-2Komplement

[http://upload.wikimedia.org/wikipedia/commons/thumb/5/5d/
4Bit-2Komplement.svg/300px-4Bit-2Komplement.svg.png](http://upload.wikimedia.org/wikipedia/commons/thumb/5/5d/4Bit-2Komplement.svg/300px-4Bit-2Komplement.svg.png)

Zuletzt: 29.04.2014

Literatur II



Jonas Fritzsch

Datentypen in C

[http://wwwlehre.dhbw-stuttgart.de/~fritzsch/Programmieren/
Folien/Teil%204%20-%20Datentypen%20in%20C.pdf](http://wwwlehre.dhbw-stuttgart.de/~fritzsch/Programmieren/Folien/Teil%204%20-%20Datentypen%20in%20C.pdf)

Zuletzt: 29.04.2014



Swanson Technologies

C Language Operator Precedence Chart

<http://www.swansontec.com/sopc.html>

Zuletzt: 29.04.2014



Wikipedia

IEEE 754

http://de.wikipedia.org/wiki/IEEE_754

Zuletzt: 29.04.2014

Literatur III



tutorialspoint

C - Constants and Literals

http://www.tutorialspoint.com/cprogramming/c_constants.htm

Zuletzt: 29.04.2014