

PR: Programmiersprache C (Linux) Einheit 3: Einführung in C

Roland A. Eggetsberger
Inst. f. Informationsverarbeitung und Mikroprozessortechnik (FIM)
Johannes Kepler Univ. Linz, Altenbergerstr. 69, A-4040 Linz, Austria

C - Grundlagen

Programmerstellung
Compilieren
Codieren
Funktionen

Programmerstellung

- Codieren

```
prog.c
```

- Compilieren

```
gcc prog.c
```

- Ausführen

```
prog (bzw. a.out)
```

Compilieren (I)

- Präprozessor
- Compiler
- Assembler
- Linker

Compilieren (II)

- Optionen (I)

```
-o ... Angabe eines Namen
-c ... Kein Linker, Object-Files können
    ohne Option gelinkt werden
-lL .. Mit Library L linkt (Option nach
    File-Argumenten angeben)
-LD .. Directory D für Object-Libraries
-IP .. Pfad P für Include-Files
```

Compilieren (III)

- Optionen (II)

```
-g ..... Debugger
-v ..... Alle Kommandos ausgeben
-Wall ... Warnungen ausgeben
-ansi ... Keine nicht-ANSI Features
```

Compilieren (IV)

- Libraries
C selbst ist sehr schlank
Eine Stärke liegt in der Fülle von Libraries
Es gibt auch Man-Pages dafür (Keywordsuche)

```
man 2 ... System Libraries
man 3 ... Sonstige
```

Codieren (I)

- Programmstruktur
Präprozessor
Typen
Funktionsprototypen
Variable
Funktionen

Codieren (II)

- Kommentar
`/* ... */`
- Was soll kommentiert werden?
Programmname - Kurzbeschreibung
Autor Genauere Beschreibung
Verwendung Referenzen
File-Formate Einschränkungen
Bearbeitungsschritte Fehlerbehandlung

Codieren (III)

- Stilparameter
Sprechende Bezeichner (mit Kommentar)
Einrückung
Klarheit (Unterteilung nach Einheiten)
Einfachheit

Funktionen (I)

- Struktur

```
/* ... */
typ name (parameter) {
    deklarationen

    anweisungen
}
```

Funktionen (II)

- Hauptprogramm

```
/* ... */
#include <stdio.h>
main () {
    deklarationen

    anweisungen
    return(0);
}
```

C - Elementare Typen

Datentypen

Char
Int
Float
Double
Wertebereiche
Ausdrücke
Konstanten

Variablen

Datentypen

• Arten

```
char
int
short
long
float
double
```

Char (I)

- Art
ASCII-Zeichen
Werden aber auch als Zahlen interpretiert

• Beispiel

```
char c;
...
c = '0';
```

Char (II)

• Spezielle Zeichen (I)

```
\b ... backspace
\f ... form feed
\n ... newline
\r ... return
\t ... tab
```

Char (III)

• Spezielle Zeichen (II)

```
\0 ... 0 (Stringende)
\' ... '
\" ... "
\\nnn . Character nnn (oktal)
```

Int (I)

- Art
Ganze Zahlen

• Länge

```
short int
long int
```

aber auch

```
short
long
```

Int (II)

- Vorzeichenlose Zahlen
Kombinationen mit `unsigned`
also z.B.

```
unsigned int
unsigned short
```

Float

- Art
Gleitkommazahlen
- Darstellung
Mathematische Exponentialdarstellung
z.B.

```
2.7E-5
```

Double

- Art
Gleitkommazahlen
- Doppelte Genauigkeit von float

Wertebereiche (I)

Type	Größe	von	Bis
char	1	-128	127
unsigned char	1	0	255
short (int)	2	-32768	32767
unsigned short	2	0	65535
long, int	4	-2^{31}	$2^{31}-1$
float	4	$3.2 \cdot 10^{38}$	$3.2 \cdot 10^{38}$
double	8	$-1.7 \cdot 10^{308}$	$1.7 \cdot 10^{308}$

Wertebereiche (II)

- Verwendung
Diese Schranken können bei manchen Systemen
auch variieren
Abruf aus `limits.h` (`/usr/include/limits.h`)

Ausdrücke

- Arten
Konstanten
Variablen
Funktionsaufrufe
Anwendungen von Operatoren auf Ausdrücke
(Ausdruck)

Konstanten (I)

- Schlüsselwort

```
const
```

- Beispiel

```
const int a = 1;
```

- Symbolische Konstanten

```
#define
```

Konstanten (II)

- Literale

```
...L ... long
...U ... unsigned
0x... ... Hexadezimal
0... ... Oktal
"Hi" ... String
```

Variablen (I)

- Name

Beginnt mit Buchstabe oder _
Sonst können auch Ziffern vorkommen
Aufpassen auf reservierte Wörter

- Deklaration

```
typ name; /* ... */
```

Variablen (II)

- Global

Vor den Funktionen
Werden initialisiert

- Lokal

In Funktionen

Variablen (III)

- Sichtbarkeit

Können verdeckt werden

- Lebensdauer

Im definierten Block
Werden nach Beenden gelöscht

C - Operatoren

```
Arithmetische Operatoren
Implizite Typkonvertierung
Vergleichsoperatoren
Logische Operatoren
Bitoperatoren
Weitere Operatoren
```

```
Auswertungsreihenfolge
I/O
```

Arithmetische Operatoren (I)

- Standard

`+, -, *, /`

- Zuweisung

`=`
`variable = ausdruck`

Zuweisungen können aber auch in Bedingungen (Testausdrücken) auftreten

Arithmetische Operatoren (II)

- Inkrement und Dekrement

`++, -- ... Präfix oder Postfix`

- Sideeffects

Beispiel

```
val = 1;
res = (val++ * 2) + (val++ * 3);
```

- Integeroperationen

`%, / ... Modulo bzw. Division`

Arithmetische Operatoren (III)

- Kurzformen

Statt

`a = a op b;`

schreibt man auch

`a op= b;`

Implizite Typkonvertierung

Operand1	Operand2	Ergebnis
int, float, double	char, short	int, float, double
double	beliebig	double
long	beliebig	long
unsigned	beliebig	unsigned
int	beliebig	int

Man beachte die Reihenfolge der Zeilen

Vergleichsoperatoren

- Gleichheit - Ungleichheit

`==, !=`

- Größer - Kleiner

`>, <`

- Größer gleich - Kleiner gleich

`>=, <=`

Logische Operatoren (I)

- Und

`&&`

- Oder

`||`

- Nicht

`!`

Logische Operatoren (II)

- Kurzschlussauswertung
Bei den zweistelligen logischen Operatoren wird in Abhängigkeit des Ergebnisses des ersten Operanden der zweite nicht mehr ausgewertet

- Beispiel

```
x = 0; y = 1;
z = x == y && y == 1;
```

Bitoperatoren (I)

- Und

&

- Oder

|

- Eor

^

Bitoperatoren (II)

- Shift (links)

<<

- Shift (rechts)

>>

- Einerkomplement

~

Weitere Operatoren (I)

- Komma-Operator

ausdr1, ausdr2 ... Wert von ausdr2

- Typkonvertierung (Cast)

(typ) ausdruck

- Adressoperator

&

Weitere Operatoren (II)

- Größenoperator

sizeof()

- Umleitungsoperator

*

- Feldindexoperator

[]

Weitere Operatoren (III)

- Elementauswahloperator

.

- Elementkennzeichnungsoperator

->

Auswertungsreihenfolge

() [] -> .	L -> R	^	L -> R
! ~ - * & sizeof (typ) ++ --	R -> L		L -> R
* / %	L -> R	&&	L -> R
+ -	L -> R		L -> R
<< >>	L -> R	? :	R -> L
< <= > >=	L -> R	= += -=	R -> L
== !=	L -> R	,	L -> R
&	L -> R		

I/O (I)

- Ausgabe

```
printf(format, variablen);
```

Man beachte dabei die Anzahl der Variablen

- Eingabe

```
scanf(format, &variablen);
```

I/O (II)

- Formate (I)

```
%c ... char
%d ... int
%f ... float (auch double)
%s ... Strings
```

I/O (III)

- Formate (II)

```
%hd ... short int
%ld ... long int
%u ... unsigned int
%hu ... unsigned short int
%lu ... unsigned long int
%lf ... double
```

C - Ablaufstrukturen

```
Anweisungen
Verzweigungen
Scheifen
I/O
```

Anweisungen

- Ausdrucksanweisung

```
ausdruck;
```

- Leere Anweisung

```
;
```

- Blöcke

```
{ }
```

Verzweigungen (I)

- If

```
if (testausdruck)
    anweisung
```

Überprüfen auf nicht Null

Verzweigungen (II)

- If - Else (I)

```
if (testausdruck)
    anweisung1
else
    anweisung2
```

Bei mehreren `if`-Statements gehört `else` immer zum letzten `if`

Besser - Klammern

Verzweigungen (III)

- If - Else (II)

```
if (testausdruck1)
    anweisung1
else if (testausdruck2)
    anweisung2
else
    anweisung3
```

Verzweigungen (IV)

- Verzweigungsoperator

```
ausdruck1 ? ausdruck2 : ausdruck3
```

Verkürzte Schreibweise, falls die Anweisungen in einer Verzweigung Ausdrucksanweisungen sind.

Verzweigungen (V)

- Vielfach (I)

```
switch (ausdruck) {
    case konstantel:
        anweisung1
        break;
    ...
    default:
        anweisung
}
```

Verzweigungen (VI)

- Vielfach (II)

Man beachte, dass `break` erforderlich ist um ein abgrenzen der Items zu ermöglichen

Schleifen (I)

- For-Schleife

```
for (ausdruck1; ausdruck2; ausdruck3)
    anweisung
```

Dabei ist

```
ausdruck1 ... Initialisierung
ausdruck2 ... Testausdruck
ausdruck3 ... Zählausdruck
```

Schleifen (II)

- While-Schleife

```
while (testausdruck)
    anweisung
```

Hier ist die Schleifensteuerung selbst zu realisieren

Schleifen (III)

- Do-Schleife

```
do
    anweisung
while (testausdruck);
```

I/O

- Zeichenweise I/O

```
c = getchar(); ... Ein Zeichen einlesen
putchar(c); ..... Ein Zeichen ausgeben
```

Literatur

- The C Programming Language; Kernighan, Brian W., Ritchie, Dennis M.; Prentice-Hall