

## KV Betriebssysteme

### Einheit 1: Einführung in C

Roland A. Eggetsberger

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

## Inhalt

- Grundlagen
- Elementare Datentypen
- Ausdrücke
- Operatoren
- Ablaufstrukturen
- I/O
- Funktionen
- Datenstrukturen

### Grundlagen

Eigenschaften  
Programmerstellung

### Eigenschaften (I)

- C ist weit verbreitet.
- C-Compiler existieren für die meisten Zielsysteme.
- C ist sehr schlank.
- C ist sehr effizient.
- Es gibt eine Fülle von Libraries.

### Eigenschaften (II)

- Es gibt in C nur Funktionen, keine Prozeduren.
- C selbst besitzt keine I/O-Anweisungen.
- Funktionen können eine variable Parameteranzahl besitzen.
- Die Parameterübergabe erfolgt durch call by value.

### Programmerstellung (I)

- Programmstruktur:
  - Präprozessor*
  - Typdeklarationen*
  - Funktionsprototypen*
  - Variablen*
  - Funktionen*
  - `main`



## Elementare Typen

Char  
Int  
Float  
Double  
Wertebereiche

## Char (I)

- ASCII-Zeichen
- Auch als Zahlen interpretierbar

Beispiel

```
char c;  
:  
i = c - '0';
```

### Char (II)

- Spezielle Zeichen

```
\b .... backspace  
\f .... form feed  
\n .... newline  
\r .... return  
\t .... tab  
\0 .... Stringende  
\' .... '  
\" .... "  
\nnn .. Character nnn (Oktal)
```

### Int (I)

- Ganze Zahlen
- Unterschiedliche Länge

```
short int i;  
long int j;
```

Aber auch

```
short k;  
long l;
```

### Int (II)

- Vorzeichenlose (ganze) Zahlen  
Kombinationen mit `unsigned`  
also z.B.

```
unsigned int i;  
unsigned short j;
```

### Float

- Gleitkommazahlen
- Darstellung:  
Mathematische Exponentialdarstellung  
z.B.

```
float x;  
x = 2.7E-5;
```

### Double

- Gleitkommazahlen
- Doppelte Genauigkeit von `float`
- Arithmetik wird in `double` betrieben

### Wertebereiche

Typ	Byte	von	bis
<code>char</code>	1	-128	127
<code>unsigned char</code>	1	0	255
<code>short</code>	2	-32768	32767
<code>unsigned short</code>	2	0	65535
<code>long, int</code>	4	$-2^{31}$	$2^{31}-1$
<code>float</code>	4	$-3.2 \cdot 10^{38}$	$3.2 \cdot 10^{38}$
<code>double</code>	8	$-1.7 \cdot 10^{308}$	$1.7 \cdot 10^{308}$
<code>void</code>	0		

### Ausdrücke

Arten von Ausdrücken

Konstante

Variablen

### Arten von Ausdrücken

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

### Konstante (I)

- Schlüsselwort

```
const
```

- Beispiel

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

- Präprozessor-Konstanten

```
#define K 2
```

### Konstante (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 (Bsp.)

```
int i, j;  
char c;
```

### Variablen (II)

- Global  
Deklaration vor den Funktionen  
Globale Variablen werden initialisiert
- Lokal  
Deklaration in einer Funktionen

### Variablen (III)

- Sichtbarkeit  
Variablen können verdeckt werden
- Lebensdauer  
Im definierten Block  
Werden nach Beenden des Blocks gelöscht

### Operatoren

Arithmetische Operatoren  
Vergleichsoperatoren  
Logische Operatoren  
Bitoperatoren  
Auswahloperator  
Weitere Operatoren  
Auswertungsreihenfolge

### Arithmetische Operatoren (I)

- Standard

`+, -, *, /`

- Zuweisung

```
int val, a, b;
val = a * b;
```

Zuweisungen können aber auch in Bedingungen (Testausdrücken) auftreten

- Inkrement und Dekrement

`++, -- ... Präfix oder Postfix`

### Arithmetische Operatoren (II)

- Sideeffects (Bsp.)

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

- Integeroperationen

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

- Kurzformen - Statt

```
a = a op b;
```

schreibt man auch

```
a op= b;
```

### Arithmetische Operatoren (III)

- Implizite Typkonvertierung

Operand	Operand	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

& .... und  
| .... oder  
^ .... eor  
<< ... Shift (links)  
>> ... Shift (rechts)  
~ .... Einerkomplement

### Auswahloperator

- Verzweigung

```
ausdruck1 ? ausdruck2 : ausdruck3
```

ausdruck1 wird ausgewertet. Falls er nicht Null ist, so wird ausdruck2 ausgewertet, andernfalls ausdruck3.

### Weitere Operatoren

,	.....	Kommaoperator
()	.....	Typkonvertierung (Cast)
&	.....	Adressoperator
sizeof()	..	Größenoperator
*	.....	Umleitungsoperator
[]	.....	Feldindexoperator
.	.....	Elementauswahloperator
->	.....	Elementkennzeichnungoperator

### Auswertungsreihenfolge

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

Assoziativität: L: ... links, R: ... rechts

### Ablaufstrukturen

Anweisungen  
Blöcke  
Verzweigungen  
Schleifen

### Anweisungen

- Ausdrucksanweisung

```
ausdruck;
```

- Leere Anweisung

```
;
```

- Blöcke

```
{ }
```

### Blöcke

- Aufbau (Bsp.)

```
i = 0;  
{  
    i = 1;  
    :  
}
```

Zusammenfassen von Anweisungen  
Variablendeklaration möglich

### Verzweigungen (I)

- `if`-Statement

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

Überprüfen auf ungleich Null.

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

Besser ist es, Blöcke zu verwenden.

### Verzweigungen (II)

- **switch-Statement**

```
switch (ausdruck) {  
    case konstante1: anweisung1; break;  
    :  
    default: anweisung;  
}
```

Man beachte, dass `break` erforderlich ist um ein abgrenzen der Fälle 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);
```

anweisung wird zumindest einmal ausgeführt.