3. Operatoren und Ausdrücke

Ausdruck (expression)

- Verarbeitungsvorschrift zur Ermittlung eines Wertes
- besteht aus Operanden und Operatoren
- wichtigste Ausdrücke: arithmetische und logische (boole'sche) Ausdrücke

Beispiele

```
int    i = 5, j = 2, k = 23;
float  x = 2.0, y = 5.5;
double d = 2.4;
```

Ausdruck	Resultat
i / j	2
k% i * j	6
k - 7 % 5	21
x*y -i	6.0
y / x	2.75
y % x	nicht erlaubt
d / 2	1.2

Operatoren und Ausdrücke (2)

Regeln

- Bei der Auswertung gelten Vorrangregeln, zum Beispiel Punktrechnung vor Strichrechnung.
- a / b ergibt für a,b int wiederum einen int-Wert, nämlich den Ganzzahl-Anteil der Division. Achtung! Es wird nicht kaufmännisch gerundet!
- a % b (die Modulo-Funktion) ist auf float und double nicht erlaubt.
- bei Mischung von int und float/double in einem Ausdruck ist das Resultat float/double.

Der L-Wert (1)

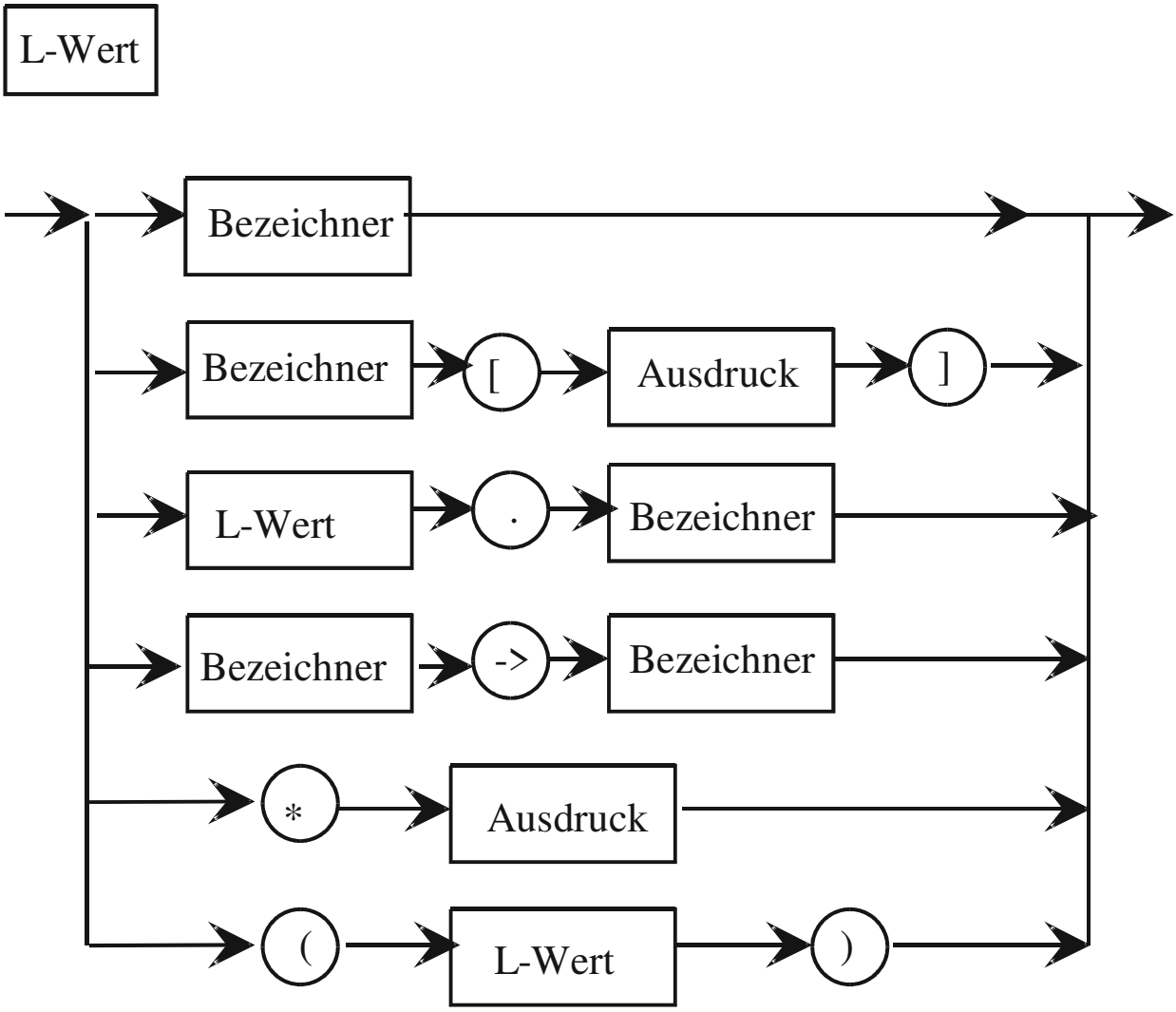
Ein L-Wert ist ein Ausdruck, der ein Objekt (einen benannten Speicherbereich) bezeichnet. L-Werte sind Objekte, denen Ergebnisse von Operationen zugeordnet werden können.

Ein Beispiel für einen L-Wert ist ein Variablenname mit geeignetem Typ und passender Speicherklasse.

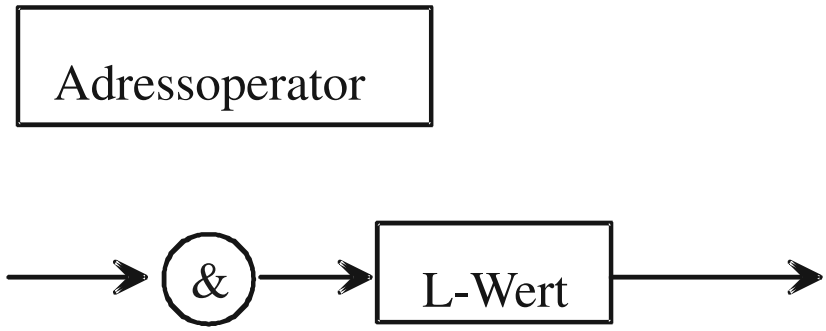
Manche Operatoren erwarten L-Werte als Operanden, manche liefern einen L-Wert als Resultat.

Vereinfacht kann man sich merken, dass ein L-Wert etwas ist, was auf der **linken** Seite einer Wertzuweisung stehen darf.

Der L-Wert (2)


L-Wert
Bezeichner
Bezeichner
[
Ausdruck
]
L-Wert
.
Bezeichner
Bezeichner
->
Bezeichner
*
Ausdruck
(
L-Wert
)

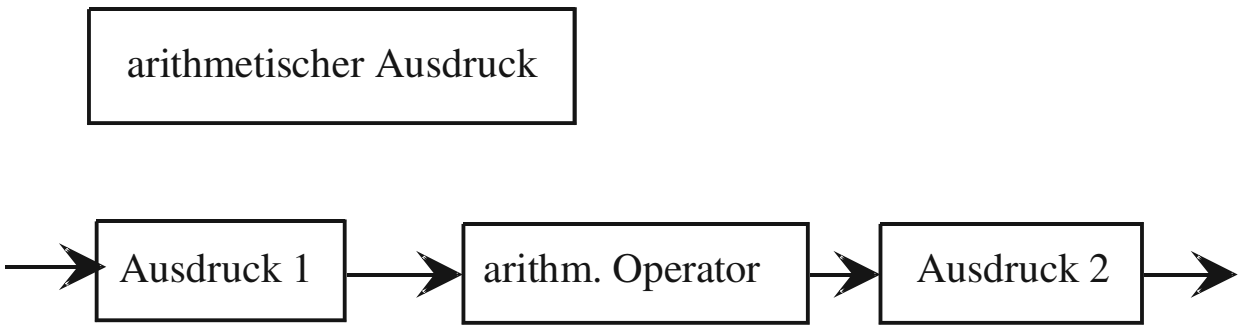
Adressoperator

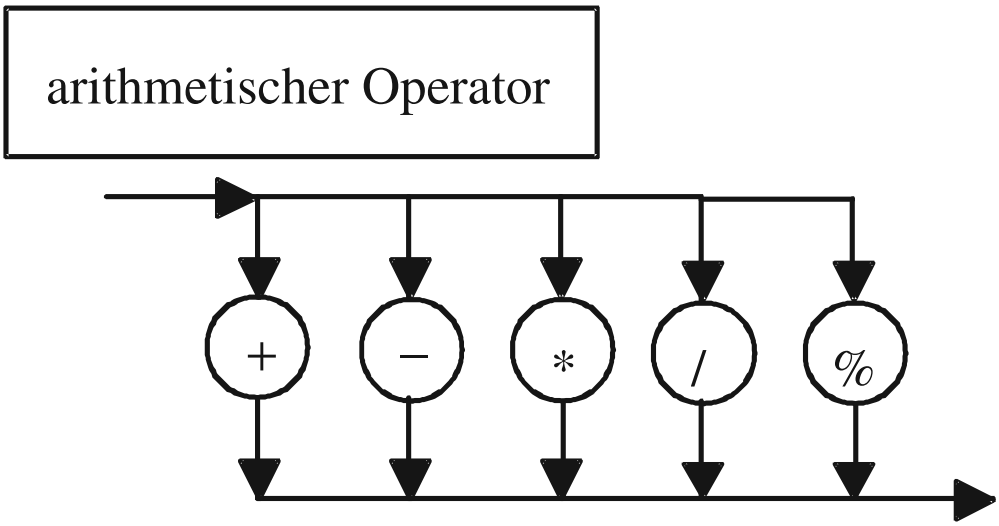


Beispiel &y

(dazu später im Kapitel über Zeiger mehr)

Arithmetische Operatoren

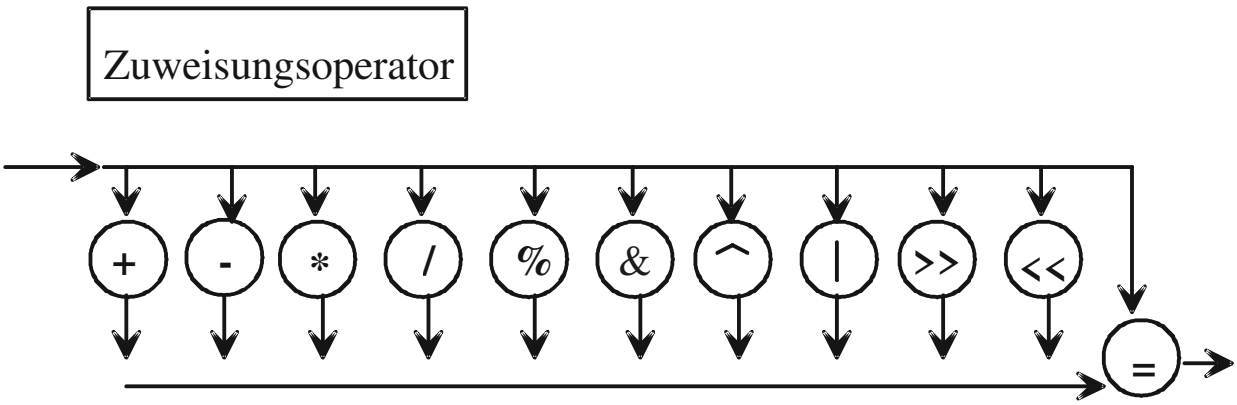




Beispiele

```
a+b
(x-y)/(z%5)
```

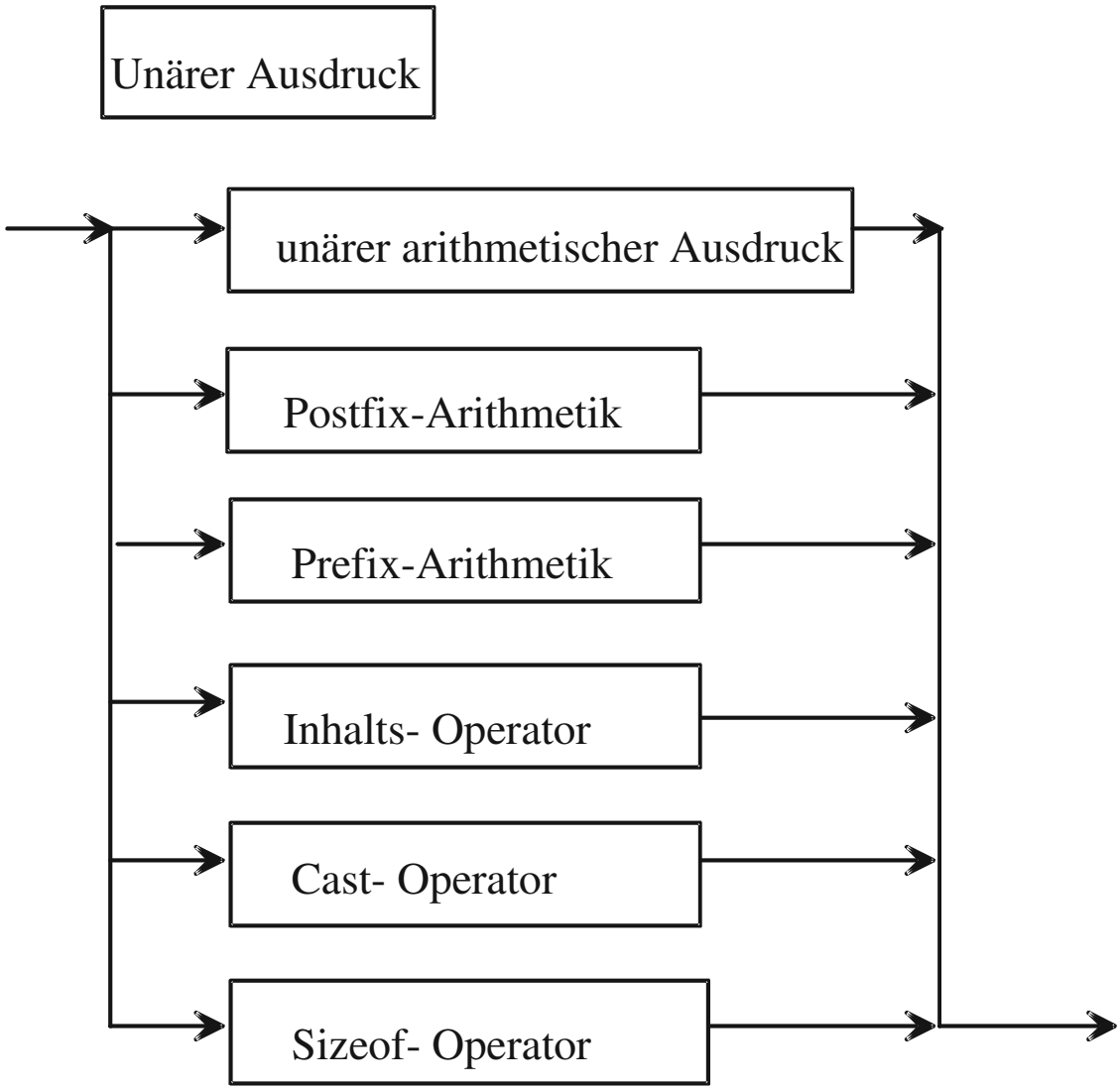
Zuweisungsoperator



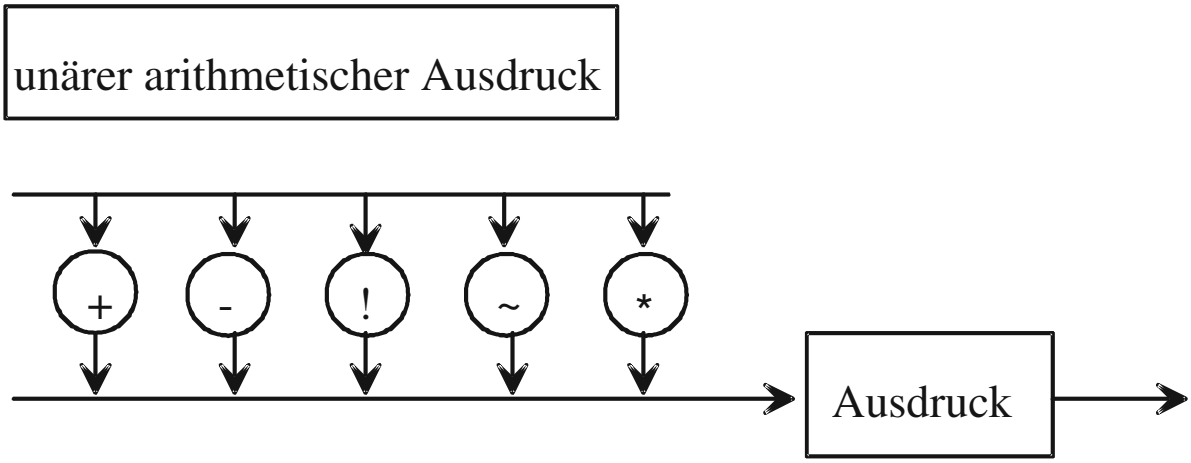
Beispiele

```
a += 5      /* dasselbe wie a = a + 5  */
l[a]  = 17 + 4
a = b = c = 7
```

Unäre Ausdrücke



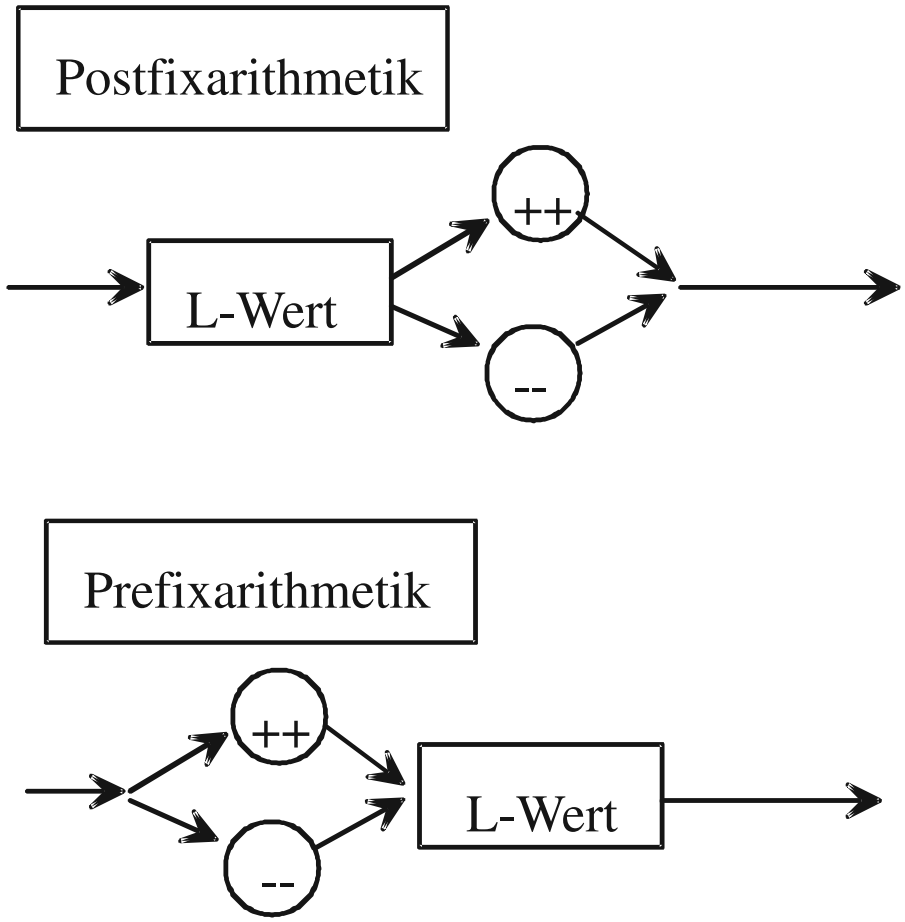
Unärer arithmetischer Ausdruck



Beispiele

`+(a+b)`	Vorzeichen
`-(x*y)`	Vorzeichen
`!a`	logische Negation (NOT)
`~a`	bitweise Negation
`*a`	Inhaltsoperator

Postfix- und Prefix-Arithmetik (1)



Postfix- und Prefix-Arithmetik (2)

Postinkrement

Der Wert der Variablen wird erst **nach** der Auswertung des Ausdrucks, in dem die Variable vorkommt, erhöht.

Preinkrement

Der Wert der Variablen wird vor der Auswertung des Ausdrucks, in dem die Variable vorkommt, erhöht.

Beispiel Postinkrement

```
int a,b;
a = 1;
b = a++;      /*  b=1, a=2  */
```

Beispiel Preinkrement

```
int a,b;
a = 1;
b = ++a;      /*  b=2, a=2  */
```

Analog für Dekrement.

Inhaltsoperator

Der Inhaltsoperator (dereferencing operator) extrahiert aus einem Speicherobjekt, dessen Adresse gegeben ist, seinen Inhalt. Er ist besonders wichtig im Zusammenhang mit der Verwendung von Zeigern. Der Operand muss vom Typ Zeiger sein.

Notation: *ausdruck

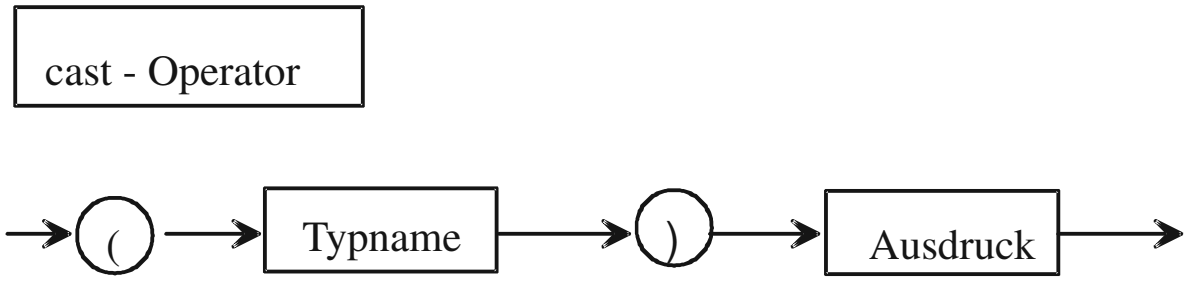
Beispiel

```
int *pi;  /*  pi ist ein Zeiger auf ein int */
float *pf;

int n;
float x;
n = 12;  x = 12.4;
pi = &n;
pf = &x;
```

`*pi` hat jetzt den Wert 12, `*pf` hat den Wert 12.4

Typkonversionsoperator (cast-Operator)

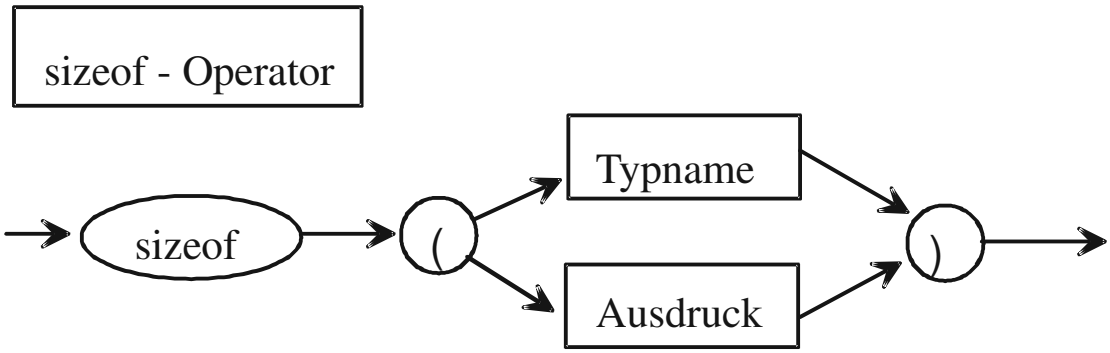


In C gibt es keine strenge Bindung der Variablen an ihre Datentypen!! Treffen in einer Operation Operanden mit unterschiedlichem Datentyp zusammen, so wandelt C nach definierten Regeln implizit um. Ist die Umwandlungsregel nicht offensichtlich (z. B. int ->float), so sollte explizit konvertiert werden, um anzuzeigen, wie die Konvertierung vom Programmierer gewollt ist.

Beispiel

```
int      a;
double   x;
a = (int) (5 * x);
```

Der sizeof-Operator



C verwendet den unären Operator `sizeof`, um die Anzahl von Bytes zu bestimmen, die zur Speicherung eines Objektes benötigt werden.

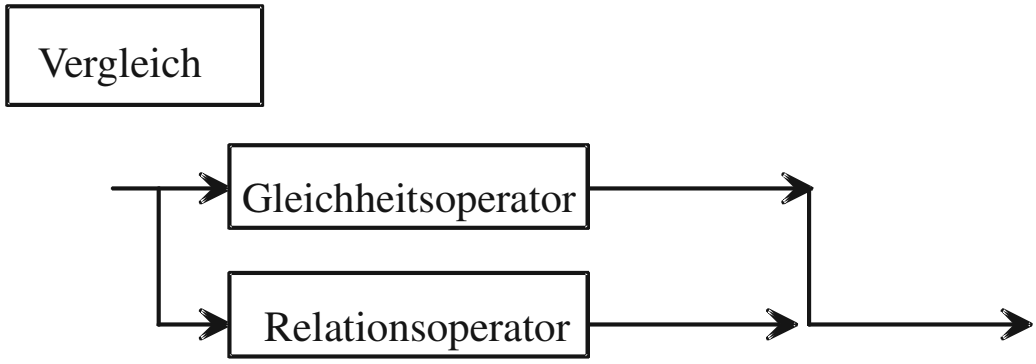
Beispiel

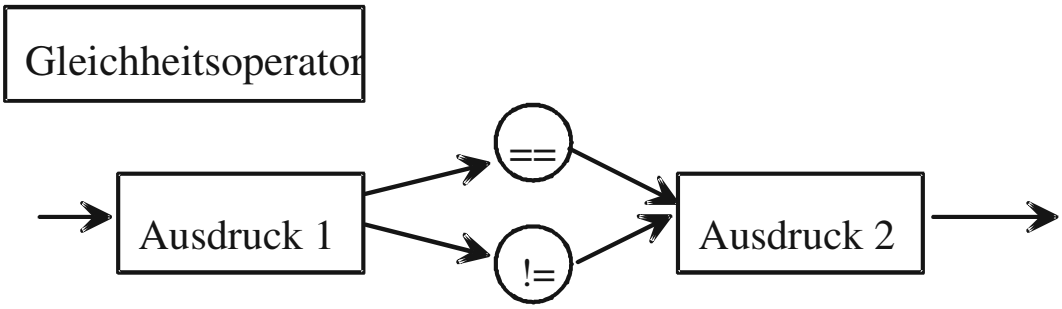
```
int a;
```

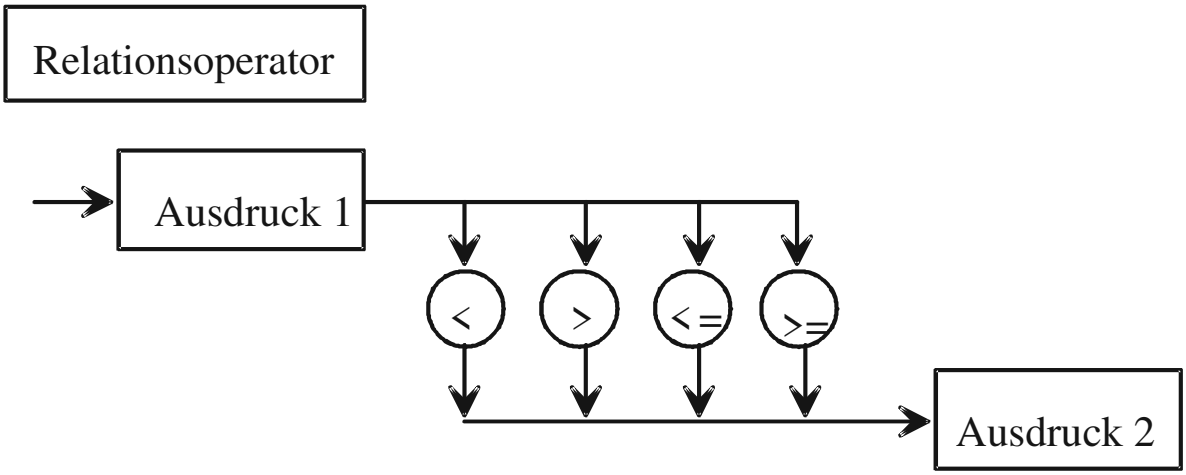
`sizeof (char)` liefert in der Regel 1

`sizeof (a*2)` liefert sizeof (int)

Vergleichsoperatoren (1)







Vergleichsoperatoren (2)

Die Rangordnung (Präzedenz) der Relationsoperatoren ist kleiner als die der arithmetischen Operatoren.

Die Rangordnung der Gleichheitsoperatoren ist kleiner als die der Relationsoperatoren.

Das Ergebnis eines Vergleichsoperators liefert entweder `falsch` oder `wahr`.

Beispiele

a == b

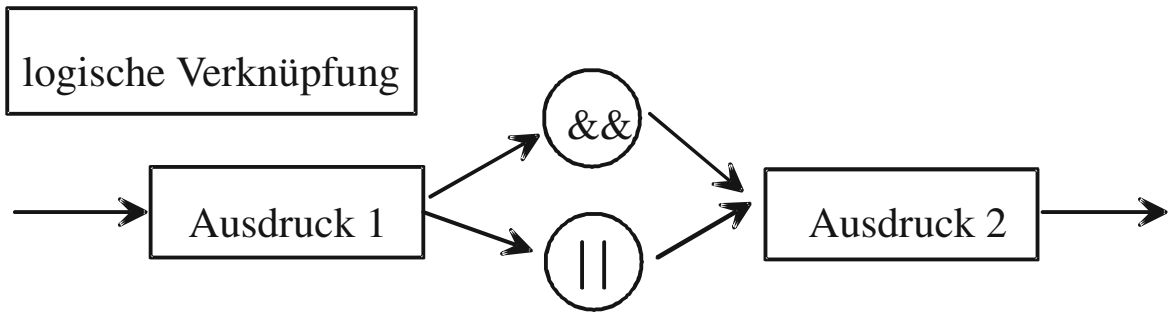
y >= z

Merke: Das Gleichheitszeichen in

a = b

ist in C der Zuweisungsoperator, nicht der Vergleichsoperator „ist gleich“!

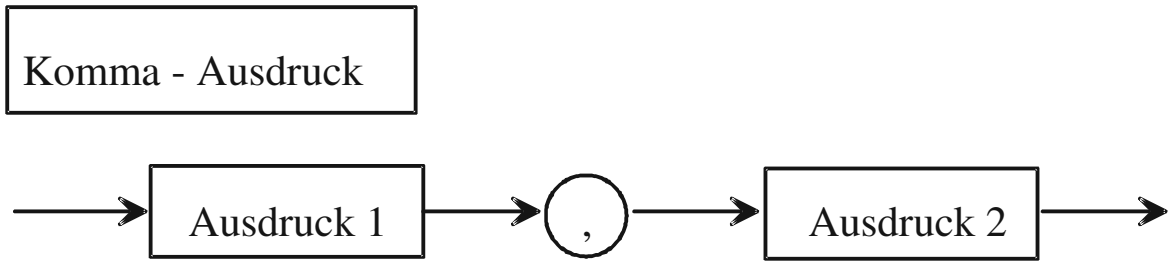
Logische Operatoren



&& ist das logische UND, | | ist das logische ODER.

Die Wertigkeit (Präzedenz) von && ist höher als die von | | , aber beide haben eine kleinere Wertigkeit als unäre, arithmetische und Vergleichsoperatoren. Lediglich die Zuweisungsoperatoren haben eine noch geringere Wertigkeit.

Der Komma-Operator



Zwei Ausdrücke werden nacheinander von links nach rechts ausgeführt.

Datentyp und Resultat des Ausdrucks sind vom Typ und Wert von Ausdruck 2.

Beispiel

```
for (s = 0, i = 0; i< n; i++)
        s += x[i];
```

summiert die Elemente des Vektors x[ ].

Spezielle Operatoren: Bitoperatoren (1)

Die kleinste adressierbare Speichereinheit in C ist ein Byte (char). Es ist aber möglich, auch auf einzelne Bits zuzugreifen. Dies geschieht mit Hilfe der Bitoperatoren.

Bitweise logische Operatoren:

Aktion	Symbol
Bitweises Komplement	~
Bitweises AND	&
Bitweises OR	|
Bitweises XOR	^

Shift-Operatoren (Linksverschiebung, Rechtsverschiebung):

LEFTSHIFT	<<
RIGHTSHIFT	>>

Spezielle Operatoren: Bitoperatoren (2)

Der Operator ~ ist als einziger **unär**. Er bildet das bitweise Komplement.

Beispiel

```
x    = (int) 5                (dezimal)
x    = 0x0000000000000101     (hexadezimal)
~x   = 0x1111111111111010     (hexadezimal)
```

Die anderen Operatoren sind **binäre** Operatoren und arbeiten bitweise, wie in der Logik definiert.

Beispiel

```
                x = 0x1100
                y = 0x1010
  =>      x ^   y = 0x0110    (XOR-Operation)
```

Achtung! Die Verwechslung von && mit & sowie von | | mit | ist ein sehr häufiger Programmierfehler in C!

Bitoperatoren (1)

Die Shift-Operatoren verschieben den gesamten Bitstring um n Bits nach rechts oder links.

Beispiel

`x = 0x0101` (= 5 dezimal)

`x << 1` ist dann 0x1010 (=10 dezimal)

Man beachte, dass durch diese Operation ein Multiplikation mit 2 stattfindet.

Bitoperatoren (2)

Beispiel für logische vs. bitweise boole'sche Operatoren

```
unsigned short int x = 2; /* Länge 1 Byte */
```

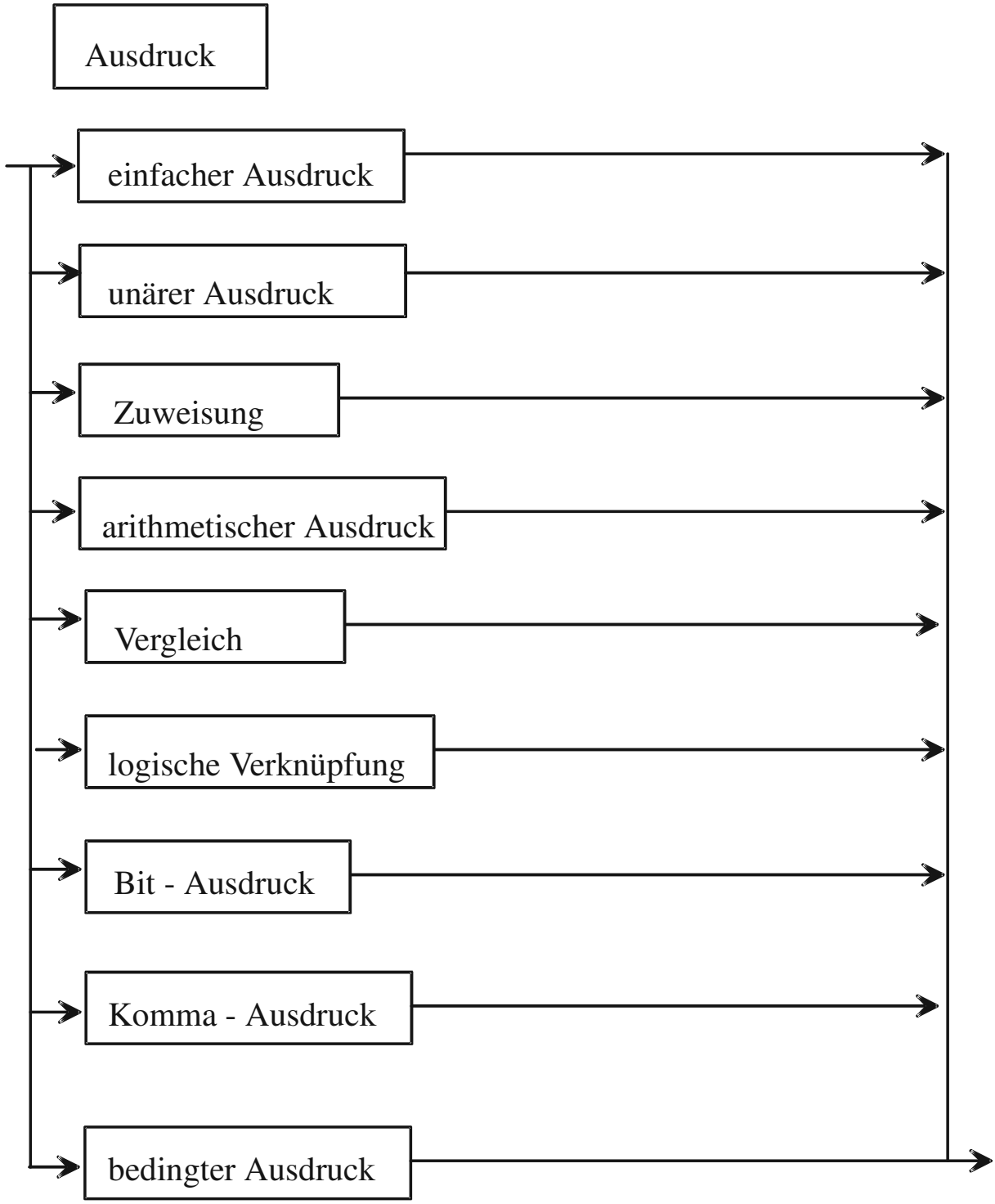
`x && 1` liefert als Ergebnis `true` zurück
(jeder Wert ≠ 0 steht für `true`)

`x & 1` liefert

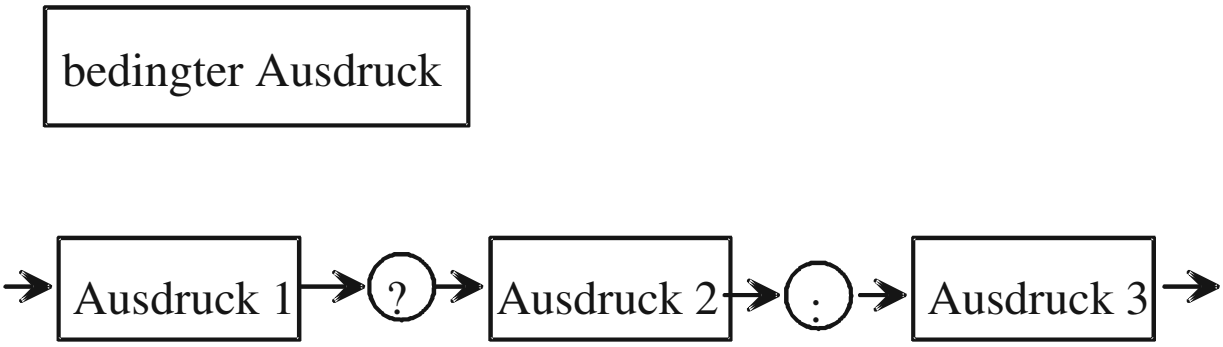
```
00000010
00000001
--------
00000000
```

Dies würde als `false` interpretiert.

Ausdruck


Ausdruck
einfacher Ausdruck
unärer Ausdruck
Zuweisung
arithmetischer Ausdruck
Vergleich
logische Verknüpfung
Bit - Ausdruck
Komma - Ausdruck
bedingter Ausdruck

Bedingter Ausdruck



Der Code

```
x = (y < z) ? y : z;
```

ist äquivalent zu

```
if (y < z)
    x = y;
else
    x = z;
```

Beispiel

```
max = (a < b) ? b : a;
```

Präzedenzregeln

Wie in der Mathematik:

- Geklammerte Ausdrücke zuerst
- Ungeklammerte Ausdrücke gemäß vier Präzedenzklassen:
  1) ! (NOT)
  2) Multiplikationsoperatoren
  3) Additionsoperatoren
  4) Vergleichsoperatoren
- Bei gleicher Präzedenz erfolgt die Abarbeitung von links nach rechts.

Beispiele

(3 <= 8 * 8 + 4) || (9 / 3 + 4 * 3 <= 10) = (3 <= 68) || (15 <= 10)
= 1 || 0
= 1 (`true`)

3 - 8 + 4 * 2 - 9 / 2 % 3 * 2 + 1 = 3 - 9 / 2 % 3 * 2 + 1
= 4 - 2
= 2

`!a && b || c` entspricht `((!a) && b) || c`

Überlauf/Unterlauf bei ganzen Zahlen (1)

Es gibt einen beschränkten Wertebereich für „int“ wegen der begrenzten Wortlänge des Computers (z. B. 32 Bits). Überlauf/Unterlauf tritt bei Verlassen dieses Wertebereiches auf.

Sei z eine ganze Zahl (also vom Datentyp int).

Dann gilt:

$$\text{min} \leq z \leq \text{max}, \quad \text{min} < \text{max}$$

wobei min und max den zulässigen Wertebereich für z begrenzen.

Damit bei arithmetischen Operationen das Ergebnis korrekt ist, müssen auch alle Zwischenresultate innerhalb des zulässigen Wertebereiches bleiben!

Überlauf/Unterlauf bei ganzen Zahlen (2)

Beispiele

Sei -min = max = 1000.

700 + 400 - 200 und
80 * 20 / 4

erzeugen Überläufe, wenn sie von links nach rechts ausgewertet werden. Solche Überläufe können durch Klammerung vermieden werden:

700 + (400 - 200)
80 * (20 / 4)

Manche Compiler kümmern sich bereits selbst um einen solche Änderung der Reihenfolge der Auswertung. Man kann sich jedoch nicht darauf verlassen!

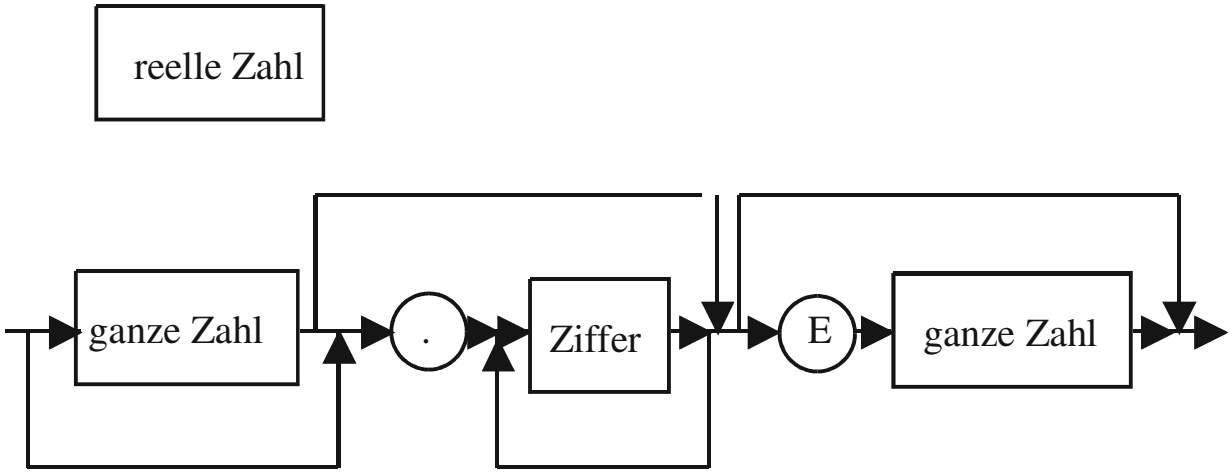
Das bedeutet auch: Das Assoziativgesetz gilt nicht mehr, wenn Bereichsgrenzen überschritten werden!

Anmerkung: Ähnliches lässt sich für das Distributivgesetz zeigen.

Überlauf/Unterlauf bei Gleitkommazahlen

Darstellung mit Mantisse und Exponent:

1.23*108 = 1.23E8
0.01 = 1.0E-2



Maschineninterne Darstellung ebenfalls mit Mantisse und Exponent. Daher gibt es zwei Arten von Bereichsgrenzen:

a) absolute Grenzen durch die Stellenzahl für den Exponenten

b) Genauigkeitsgrenzen durch die Stellenzahl für die Mantisse

Beispiel:
```
a = sqrt(2.0);      a,b reell
b = a*a;            /* b = 1.9999998 */
```

Operatoren (1)

Rang	Art	Symbol in der Logik/ Arithmetik	Operator in C	Name	Beispiel
1		( )	( )	Klammern	(a+b)
1		[ ]	[ ]	Array/ Vektor	a [ ]
1	monadisch		->	Komponente	a -> b
1	monadisch		.	Komponente	a . b
2	monadisch		++ --	Addition von 1 Subtraktion von 1	a ++ -- b
2	monadisch/ logisch	¬	!	Negation	!true
2	monadisch/ logisch	NOT	~	Bitkomplement	~y
2	monadisch		(type)	typecast	(int) x
2	monadisch		sizeof	sizeof	sizeof(int)
2	monadisch/ arithmetisch	+ -	+ -	Plusvorzeichen Minusvorzeichen	+7 -7
2	monadisch		*	Verweis	*a
2	monadisch		&	Adresse	&a
3	dyadisch/ arithmetisch/ multiplikativ	* / DIV MOD	* / / %	Multiplikation Division ganzz. Division ganzz. Rest	3 * 4 3 / 4 3 / 4 = 0 3%4 = 3
4	dyadisch/ arithmetisch/ additiv	+ -	+ -	Addition Subtraktion	3 + 4 3 - 4

Operatoren (2)

5	dyadisch/ logisch	LSHIFT	<<	Linksschift um Bitpositionen	x<<5
5	dyadisch/ logisch/	RSHIFT	>>	Rechtsshift um Bitpositionen	x>>5
6	dyadisch/ logisch/	< ≤ > ≥	< <= > >=	Relations-operatoren	a > b
7	dyadisch	== =!	== =!	Gleichheits-operatoren	a == b
8	dyadisch/ logisch/ multiplikativ	∧	&	logisches 'und' auf Bits	x & y
9	dyadisch/ logisch/ multiplikativ	XOR	∧	logisches 'XOR' auf Bits	x^y
10	dyadisch/ logisch/ additiv	∨	|	logisches 'oder' auf Bits	x | y
11	dyadisch/ logisch/ multiplikativ	∧	&&	logisches 'und'	a && b

Operatoren (3)

12	dyadisch/ logisch/ additiv	$\vee$	||	logisches 'oder'	a || b
13			?:	bedingter Ausdruck	a ? c : b
14			= += -= *= /= %= &= ^= |= <<= >>=	Zuweisungs- operatoren	a += 1 a = a + 1
15			,	Kommaoperator	s = 0, i = 0