Operatoren in C

Prioritätenliste der Auswertung

Die Operatoren mit niedrigerer Gruppen-Nummer werden zuerst ausgewertet, binden also am stärksten

Gruppe	Operator	Beschreibung	Beispiel	
1	()	Funktionsklammer	sin(a * b)	//s. Kap. 10
	[ ]	Vektorklammer	a[i]	//s. Kap. 11
	->	Strukturpointer	adr->strasse	//s. Kap. 11
	.	Strukturselektor	mitarb.vname	//s. Kap. 11
2	!	Negationsoperator	!x	//liefert 1 oder 0
	~	Komplementoperator	~b	//kippt *b* bitweise um
	++	Incrementoperator	++i	//vor Verwendung erhöhen
			i++	//nach Verwendung erhöhen
	--	Dekrementoperator	--i	//vor Verwendung //vermindern
			i--	//nach Verwendung verm.
	-	neg. Vorzeichen	-z	//neg. Wert von *z*
	(typ)	Cast-Operator	(int) ausdr	//Wert des Ausdrucks wird //nach `int` umgewandelt
	*	Inhaltsoperator	*p	//Inhalt des Pointers *p*
	&	Adressoperator	&a	//Adresse der Variablen *a*
	sizeof	Größenoperator	sizeof(x)	//Größe der Variablen *x* //in Byte
			sizeof(int)	//Größe von `int` in Byte
3	*	Multiplikationsop.	x * y	
	/	Divisionsoperator	a / 4.7	
	%	Modulo-Operator	z % 7	//0 wenn *z* / 7 = 0
4	+	Additionsoperator	b + 3	
	-	Subtraktionsop.	z - y	
5	<<	Links-Shift-Op.	a << 3	//*a* um 3 Bits nach links //schieben
	>>	Rechts-Shift-Op.	b >> 2	//*b* um 2 Bits nach rechts //schieben
6	<	kleiner-Operator	x < 5	//1 wenn wahr
	<=	kleiner-gleich-Op.	x <= 5	//1 wenn wahr
	>	größer-Operator	y > 5	//1 wenn wahr
	>=	größer-gleich-Op.	y >= 5	//1 wenn wahr
7	==	Gleichheitsoperator	a == b	//1 wenn wahr
	!=	Ungleichheitop.	z != 7	//1 wenn wahr
8	&	bitweiser UND-Op.	a & 0x7	// s. Teil III

9	^	bitweiser XOR-Op.	b ^ x	// s. Teil III
10	|	bitweiser ODER-Op.	c | 0x4	// s. Teil III
11	&&	logischer UND-Op.	x && y	//1 wenn beide nicht 0
12	||	logischer ODER-Op.	a || b	//1 wenn mind. ein Operand //ungleich 0
13	?:	bedingter Bewertungs-Op	a ? b : c	// liefert *b* wenn. *a* ungleich 0 //(wahr), sonst *c*
14	=	Zuweisungsoperator	y = a / b * 4.76	
	+=	Zuweisungsoperator	a += b + 4	//*a* = *a* + (*b* + 4)
	-=	Zuweisungsoperator	a -= b + 4	//*a* = *a* - (*b* + 4)
	*=	Zuweisungsoperator	a *= b + 4	//*a* = *a* * (*b* + 4)
	/=	Zuweisungsoperator	a /= b + 4	//*a* = *a* / (*b* + 4)
	%=	Zuweisungsoperator	a %= b + 4	//*a* = *a* % (*b* + 4)
	>>=	Zuweisungsoperator	a >>= 1	//*a* = *a* >> 1
	<<=	Zuweisungsoperator	a <<= 2	//*a* = *a* << 2
	&=	Zuweisungsoperator	a &= 0xdf	//*a* = *a* & 0xdf
	|=	Zuweisungsoperator	a |= 0xa8	//*a* = *a* | 0xa8
	^=	Zuweisungsoperator	a ^= 0x3e	//*a* = *a* ^ 0x3e
15	,	Folgeoperator	x = (i++, a + 4)	//zuerst wird *i* erhöht, dann //wird *a* + 4 berechnet; der //letzte Ausdruck (hier: *a* + 4) //wird x zugewiesen

Wenn mehrere Operatoren derselben Gruppe in einem Ausdruck aufeinanderfolgen, erfolgt die Abarbeitung entweder von links nach rechts oder von rechts nach links, je nach Gruppe:

Gruppe	Operatoren	Reihenfolge der Abarbeitung
1	() [ ] -> .	links nach rechts →
2	! ~ ++ -- - (typ) * & sizeof	rechts nach links ←
3	* / %	links nach rechts →
4	+ -	links nach rechts →
5	<< >>	links nach rechts →
6	< <= >= >	links nach rechts →
7	== !=	links nach rechts →
8	&	links nach rechts →
9	^	links nach rechts →
10	|	links nach rechts →
11	&&	links nach rechts →
12	||	links nach rechts →
13	?:	rechts nach links ←
14	= += -= *= /= %= >>= <<= &= |= ^=	rechts nach links ←
15	,	links nach rechts →

(aus Küveler & Schwoch 1999, S. 86 ff.)