Operatorer och satser i C++

Operatorer

Tabellen nedan visar operatorerna i C++, ordnade i sjunkande *prioritet*. Kolumnen längst till höger anger *associativiteten* för respektive operator, *V* anger vänsterassociativitet, *H* anger högerassociativitet.

Operator	Funktion	Assoc
klassnamn `::` *medlem*	klassmedlem	*V*
namnrymdsnamn `::` *medlem*	namnrymdsmedlem	*V*
`::` *namn*	globalt namn	*H*
`::` *kvalificerat-namn*	globalt namn	*H*
objekt `.` *medlem*	medlemselektion i **struct** eller **class**	*V*
pekare `->` *medlem*	medlemselektion i **struct** eller **class**	*V*
pekare `[` *uttryck* `]`	indexering av fält	*V*
uttryck `(` *uttryckslista* `)`	funktionsanrop	*V*
enkel-typ `(` *uttryckslista* `)`	värdekonstruktion (typomvandling)	*V*
lvalue `++`	postfix uppstegning	*H*
lvalue `--`	postfix nedstegning	*H*
typeid `(` *type* `)`	typidentifiering	*H*
typeid `(` *uttryck* `)`	dynamisk typidentifiering	*H*
dynamic_cast`<`*typ*`>(`*uttryck*`)`	dynamisk typomvandling	*H*
static_cast`<`*typ*`>(`*uttryck*`)`	statisk typomvandling	*H*
reinterpret_cast`<`*typ*`>(`*uttryck*`)`	okontrollerad typomvandling	*H*
const_cast`<`*typ*`>(`*uttryck*`)`	konstantomvandling	*H*
sizeof *uttryck*	objektstorlek	*H*
sizeof `(` *typ* `)`	typstorlek	*H*
`++`*lvalue*	prefix uppstegning	*H*
`--`*lvalue*	prefix nedstegning	*H*
`~`*uttryck*	komplement	*H*
`!` *uttryck*	logisk negation (icke)	*H*
`+`*uttryck*	enställigt (unärt) plus	*H*
`-`*uttryck*	enställigt (unärt) minus	*H*
`&`*lvalue*	adresstagning	*H*
`*`*uttryck*	avreferering av pekare	*H*
new *typ*	tilldela dynamiskt minne	*H*
new *typ* `(` *uttrycklista* `)`	tilldela dynamiskt minne och initiera	*H*
new `(` *uttryckslista* `)` *typ*	*placement syntax*; extra arg. till **new**	*H*
new `(` *uttryckslista* `)` *typ* `(` *uttryckslista* `)`	*placement syntax* och initiering	*H*
delete *pekare*	återlämna dynamiskt minne	*H*
delete `[ ]` *pekare*	återlämna dynamiskt minne för fält	*H*
`(` *typ* `)` *uttryck*	statisk typomvandling (”cast”)	*H*
objekt `.*` *pekare-till medlem*	pekare-till-medlem-selektion	*V*
pekare `->*` *pekare-till medlem*	pekare-till-medlem-selektion	*V*

Operator	**Funktion**	**Assoc**
uttryck * *uttryck*	multiplikation	*V*
uttryck / *uttryck*	division	*V*
uttryck % *uttryck*	modulo (rest vid heltalsdivision)	*V*
uttryck + *uttryck*	addition	*V*
uttryck - *uttryck*	subtraktion	*V*
uttryck << *uttryck*	bitvis vänsterskift (nollutfyllnad)	*V*
uttryck >> *uttryck*	bitvis högerskift (aritmetiskt, logiskt?)	*V*
uttryck < *uttryck*	mindre än	*V*
uttryck <= *uttryck*	mindre än eller lika med	*V*
uttryck > *uttryck*	större än	*V*
uttryck >= *uttryck*	större än eller lika med	*V*
uttryck == *uttryck*	lika med	*V*
uttryck != *uttryck*	ej lika med	*V*
uttryck & *uttryck*	bitvis OCH	*V*
uttryck ^ *uttryck*	bitvis exklusivt ELLER	*V*
uttryck | *uttryck*	bitvis (inklusivt) ELLER	*V*
uttryck && *uttryck*	logiskt OCH	*V*
uttryck || *uttryck*	logiskt ELLER	*V*
lvalue = *uttryck*	enkel tilldelning	*H*
lvalue *= *uttryck*	multiplicera och tilldela	*H*
lvalue /= *uttryck*	dividera och tilldela	*H*
lvalue %= *uttryck*	modulo och tilldela	*H*
lvalue += *uttryck*	addera och tilldela	*H*
lvalue -= *uttryck*	subtrahera och tilldela	*H*
lvalue <<= *uttryck*	skifta vänster och tilldela	*H*
lvalue >>= *uttryck*	skifta höger och tilldela	*H*
lvalue &= *uttryck*	bitvis OCH och tilldela	*H*
lvalue ^= *uttryck*	bitvis exklusivt ELLER och tilldela	*H*
lvalue |= *uttryck*	bitvis (inklusivt) ELLER och tilldela	*H*
uttryck ? *uttryck* : *uttryck*	villkorsuttryck	*V*
throw *uttryck*	*throw*-uttryck (har typen **void**)	
uttryck, *uttryck*	kommauttryck	*V*

Associativiteten bestämmer beräkningordningen en för deluttryck i sammansatta uttryck där operatorerna har lika prioritet och prioriteten därför inte kan bestämma beräkningsordningen för deluttrycken. För t.ex. de additiva operatorerna (+ och -) gäller vänsterassociativetet, vilket betyder att ett uttryck som *a*+*b*-*c* beräknas från vänster till höger, dvs som om parenteser enligt (*a*+*b*)-*c* fanns. Detta betyder också för relationsoperatorerna att t.ex. *a*<*b*<*c* innebär (*a*<*b*)<*c*, inte (*a*<*b*) && (*b*<*c*).

Beräkningsordningen för operander och deluttryck är för de flesta operatorer ej definierad, vilket innebär att man för ett uttryck som t.ex. *a*+*b* inte kan säga om *a* beräknas före *b*, eller tvärt om. Detta kan ha betydelse om beräkningen av operander och deluttryck har sidoeffekter. Om t.ex. *a* är ett funktionsanrop och *b* en variabel och funktionen har sidoeffekten att ändra värdet på *b*, har beräkningsordningen betydelse. Operatorerna för tilldelning och upp- och nedstegning har sidoeffekt, genom att de ändrar värdet på sin operand. (vänsteroperanden i fallet tilldelning).

För några operatorer finns en bestämd beräkningsordning då det gäller deras operander. Dessa är logiskt OCH (&&), logiskt ELLER (||), villkorsoperatorn (?:), samt kommaoperatorn (,):

- Operatorerna && och || är s.k. *kortslutande*. I båda fallen beräknas alltid vänsteroperanden först. Om vi har *a*&&*b* och *a* beräknas till *falskt* beräknas inte b, eftersom hela uttryckets värde då måste vara *falskt*. Om vi har *a*||*b* och *a* beräknas till *sant*, måste hela uttryckets värde vara *sant* och i det fallet beräknas inte *b.*
- För villkorsoperatorn, *a*?*b*:*c*, gäller att först beräknas *a*. Om *a* blir *sant*, beräknas *b* och detta värde blir villkorsuttryckets värde. I annat fall beräknas *c* och dess värde blir villkorsuttryckets värde.
- För kommaoperatorn, *a*, *b*, gäller att först beräknas *a* och sedan *b* och dess värde blir också kommauttryckets värde.

Begreppet *lvalue*, som anges som operand för en del operatorer i prioritetstabellen, innebär att operanden ska referera till ett objekt eller en funktion. *lvalue* är en förkortning av *left value*, vilket kan tolkas som att operanden ska vara något som kan stå till vänster i en tilldelning (adressen till det minne som ska tilldelas värdet). Detta gäller förstås alla tilldelningsoperatorers vänsteroperand, men för övrigt även operanden till stegningsoperatorerna ++ och --, vilka ändrar värdet på sin operand (i princip en sido-effekt men vanligtvis den huvudeffekten man är ute efter). Motsatsen till *lvalue* är *rvalue*.

Alla operatorer, utom ::, ., .*, och ?:, är överlagringsbara.

Satser

Nedan följer syntaxen för satserna i C++. Beskrivningen är i viss mån förenklad och inte heller fullständigt beskriven i detalj. Syntaktiska element markerade med $_{opt}$ är valfria och kan utelämnas.

Syntax	**Kommentar**
sats :	en sats avslutas alltid av ett semikolon
etiketterad-sats	sats med någon form av prefix
uttryckssats	ett uttryck följt av semikolon blir en sats
sammansatt-sats	även kallad *blocksats*
selektionssats	val
iterationssats	även kallad *repetitionssats*
hoppsats	bryter den sekventiella exekveringen
deklarationssats	även deklarationer räknas som satser
try-block	för infångande av undantag
etiketterad-sats :	
identifierare : *sats*	enda användning är som mål för **goto**
case *konstant-uttryck* : *sats*	läge i **switch**
default : *sats*	valfritt, sista läge i **switch**
uttryckssats :	
$uttryck_{opt}$;	enbart semikolon kallas *tom sats*
sammansatt-sats :	satsblock, vanligtvis bestående av flera satser
{ $satssekvens_{opt}$ }	blockparentes, kan vara tom.
selektionssats :	
if (*villkor*) *sats*	*sats* utförs om *villkor* beräknas till SANT
if (*villkor*) *sats* **else** *sats*	*sats* efter **else** utförs då *villkor* FALSKT
switch (*villkor*) *sats*	*sats* är normalt en sammansatt sats
villkor :	*Integral* el. **enum** för **switch**, övriga **bool**
uttryck	av typ **bool**, *integral* eller **enum**
typspecificerare deklarator = *uttryck*	räckvidden är de undersatser *villkor* styr över

iterationssats :	
while (*villkor*) *sats*	förtestad, villkorsstyrd repetition
do *sats* **while** (*villkor*) ;	eftertestad, villkorsstyrd repetition
for (*for-initieringssats* $villkor_{opt}$; $uttryck_{opt}$) *sats*	förtestad, villkorsstyrd repetition
for-initieringssats :	*anm*. Avslutas med semikolon
uttryckssats	t.ex. i = 0;
enkel-deklaration	t.ex. **int** i = 0;
hoppsats :	
break ;	hopp ur iterationssats eller **switch**
continue ;	hopp till nästa iteration i iterationssats
return $uttryck_{opt}$;	retur från funktion, *uttryck* anger returvärde
goto *identifierare* ;	hopp till etiketterad sats
deklarationssats :	
blockdeklaration	deklaration som kan förekomma i block.
blockdeklaration :	
enkel-deklaration	t.ex. en variabeldeklaration
asm-deklaration	**asm** (*stränglitteral*) ;
namnrymdalias-deklaration	**namespace** *identifierare* = …
using-deklaration	**using** *typnamn* :: *identifierare*
using-direktiv	**using namespace** …
try-block : [a]	
try { $satssekvens_{opt}$ } **catch** (*undantagsdeklaration*) { $satssekvens_{opt}$ }	
try *sammansatt-sats* *hanterarsekvens*	mer formell syntax än ovanstående exempel.
hanterarsekvens :	
hanterare $hanterarsekvens_{opt}$	
hanterare :	
catch (*undantagsdeklaration*) *sammansatt-sats*	kan finnas en eller flera till ett **try**-block
undantagsdeklaration :	
typspecifierare *deklarator*	t.ex. **const** exception& e
typspecifierare	t.ex. **const** exception&
...	endast som sista hanterare, fångar allt.

a. Det finns även en konstruktion som kallas *funktion-try-block*, som kan förekomma i en konstruktor och där används för att fånga undantag som genereras i konstruktorns *initierare*. Det är dock ingen sats.

Det finns en hel del kompletterande semantiska regler för satserna, t.ex. vad *villkor* kan vara i olika satser där det förekommer.