Precedence

Last updated 8/20/20

Precedence

- These slides introduce precedence
- Upon completion: You should be able interpret expressions and code based on precedence

Precedence

- Precedence
  - Order in which operators are evaluated
    - In math: * and / before + and –
    - 2/3+3*4 → ((2/3) + (3*4))
- Associativity
  - Order in which operators with the same precedence are evaluated
    - In math: left to right
    - 2 + 3 – 4 + 5 → (((2 + 3) – 4) + 5)

Precedence

Precedence	Operator	Description	Associativity
1	++ --	Suffix/postfix increment and decrement	Left-to-right
	()	Function call	
	[]	Array subscripting	
	.	Structure and union member access	
	->	Structure and union member access through pointer	
	(*type*){*list*}	Compound literal(C99)	
2	++ --	Prefix increment and decrement	Right-to-left
	+ -	Unary plus and minus	
	! ~	Logical NOT and bitwise NOT	
	(*type*)	Type cast	
	*	Indirection (dereference)	
	&	Address-of	
	sizeof	Size-of	
	_Alignof	Alignment requirement(C11)	
3	* / %	Multiplication, division, and remainder	Left-to-right
4	+ -	Addition and subtraction	
5	<< >>	Bitwise left shift and right shift	
6	< <=	For relational operators < and ≤ respectively	
	> >=	For relational operators > and ≥ respectively	
7	== !=	For relational = and ≠ respectively	
8	&	Bitwise AND	
9	^	Bitwise XOR (exclusive or)	
10	|	Bitwise OR (inclusive or)	
11	&&	Logical AND	
12	||	Logical OR	
13	?:	Ternary conditional	Right-to-Left
14	=	Simple assignment	
	+= -=	Assignment by sum and difference	
	*= /= %=	Assignment by product, quotient, and remainder	
	<<= >>=	Assignment by bitwise left shift and right shift	
	&= ^= |=	Assignment by bitwise AND, XOR, and OR	
15	,	Comma	Left-to-right

Precedence

Precedence	Operator	Description	Associativity
1	++ --	Suffix/postfix increment and decrement	Left-to-right
	()	Function call	
	[]	Array subscripting	
	.	Structure and union member access	
	->	Structure and union member access through pointer	
	(*type*){*list*}	Compound literal(C99)	
2	++ --	Prefix increment and decrement	Right-to-left
	+ -	Unary plus and minus	
	! ~	Logical NOT and bitwise NOT	
	(*type*)	Type cast	
	*	Indirection (dereference)	
	&	Address-of	
	sizeof	Size-of	
	Alignof	Alignment requirement(C11)	
3	* / %	Multiplication, division, and remainder	Left-to-right
4	+ -	Addition and subtraction	
5	<< >>	Bitwise left shift and right shift	
6	< <=	For relational operators < and ≤ respectively	
	> >=	For relational operators > and ≥ respectively	
7	== !=	For relational = and ≠ respectively	
8	&	Bitwise AND	
9	^	Bitwise XOR (exclusive or)	
10	|	Bitwise OR (inclusive or)	
11	&&	Logical AND	
12	||	Logical OR	
13	?:	Ternary conditional	Right-to-Left
14	=	Simple assignment	
	+= -=	Assignment by sum and difference	
	*= /= %=	Assignment by product, quotient, and remainder	
	<<= >>=	Assignment by bitwise left shift and right shift	
	&= ^= |=	Assignment by bitwise AND, XOR, and OR	
15	,	Comma	Left-to-right

* & and () have multiple definitions
Usage is context dependent

Precedence

- Examples (ints)

a = 2, b=3, c=4

1 + 2 * 3 →

1 + 2 * 3 / 2 →

-b++ →

a += b *= c -= 3 →

--a * (1 + b) / 3 – c++ * b →

Precedence

- Examples

a = 2, b=3, c=4

1 + 2 * 3 → 1 + (2 * 3) = 7

1 + 2 * 3 / 2 → 1 + ((2 *3) /2) = 1 + (6/2) = 4
same precedence (L-R)

-b++ → -(b++) = -3 evaluates first - (b is now 4)

a += b *= c -= 3 → c= 1, b=3, a=5 same precedence (R-L)

--a * (1 + b) / 3 – c++ * b →

--a * (1 + b) / 3 – c++ * b
--a * 4 / 3 – c++ * b
--a * 4 / 3 – 4 * b
1 * 4 / 3 – 4 * b
4 / 3 – 4 *3
1 - 12
-11

Precedence

- Precedence and Associativity

For clarity and precision

Use Parenthesis freely

a = 2, b=3, c=4

(((--a) * (1 + b))/ 3) – ((c++) * b) $\rightarrow$

(((1) * (4)) / 3) – ((4) * 3)
((4 / 3) – (12))
(1 – 12)
-11