Lecture 02.02 Exploring C—operator precedence and associativity

Table 02.1 lists all C operators in order of their precedence (highest to lowest). Operators within the same box have equal precedence.

Note 1—Postfix increment/decrement have high precedence, but the actual increment or decrement of the operand is delayed (to be accomplished sometime before the statement completes execution). So in the statement `y = x * z++;` the current value of z is used to evaluate the expression (i.e., `z++` evaluates to `z`) and `z` only incremented after all else is done.

02.02.1 Operator precedence

When an expression contains two or more operators, normal operator precedence rules are applied to determine the order of evaluation. If two operators have different levels of precedence, the operator with the highest precedence is evaluated first. For example, multiplication is of higher precedence than addition, so the expression `2+3*4` is evaluated as

```
3 * 4 // = 12
2 + 12 // = 14
```

The evaluation order can be explicitly controlled using parentheses; e.g., `(2+3)*4` is evaluated as

```
2 + 3 // = 5
5 * 4 // = 20
```

Operators in Table 02.1 are grouped from highest to lowest precedence.

02.02.2 Operator associativity

If two operators in an expression have the same precedence level, they are evaluated from left to right or right to left depending on their associativity. For example, addition's associativity is left-to-right, so the expression `2+3+4` is evaluated as `(2+3)+4`. In contrast, the assign operator's associativity is right-to-left; so the expression `x=y=z` is evaluated as `x=(y=z)`.

REVIEW DRAFT

Table 02.1: C operator precedence and associativity.

Operator	Description	Associativity
`()`	Parentheses (grouping)	left-to-right
`[]`	Brackets (array subscript)	
`.`	Member selection via object name	
`->`	Member selection via pointer	
`++ --`	Postfix increment/decrement (see Note 1)	
`++ --`	Prefix increment/decrement	right-to-left
`+ -`	Unary plus/minus	
`! ~`	Logical negation/bitwise complement	
`(type)`	Cast (change type)	
`*`	Dereference	
`&`	Address	
`sizeof`	Determine size in bytes	
`* / %`	Multiplication/division/modulus	left-to-right
`+ -`	Addition/subtraction	left-to-right
`<< >>`	Bitwise shift left, Bitwise shift right	left-to-right
`< <=`	Relational less than/less than or equal to	left-to-right
`> >=`	Relational greater than/greater than or equal to	
`== !=`	Relational is equal to/is not equal to	left-to-right
`&`	Bitwise AND	left-to-right
`^`	Bitwise exclusive OR	left-to-right
`|`	Bitwise inclusive OR	left-to-right
`&&`	Logical AND	left-to-right
`||`	Logical OR	left-to-right
`?:`	Ternary conditional	right-to-left
`=`	Assignment	right-to-left
`+= -=`	Addition/subtraction assignment	
`*= /=`	Multiplication/division assignment	
`%= &=`	Modulus/bitwise AND assignment	
`^= |=`	Bitwise exclusive/inclusive OR assignment	
`<<= >>=`	Bitwise shift left/right assignment	
`,`	Comma (separate expressions)	left-to-right

REVIEW DRAFT