24 Operator precedence

We know the BODMAS rules for arithmetic operators. C is full of operators, and they have carefully defined 'precedence.'

- The highest precedence operators are evaluated left-to-right. Otherwise they have equal precedence. In this and other groups, where 'right to left' or 'left to right' is stated, this fixes the precedence where otherwise they have equal precedence.

  `LR: [] . -> postfix ++ --`

  - [] (i.e., accessing array element)
  - . (structure member)
  - $->$ (structure member through pointer)
  - Postfix increment/decrement

- Next, ***right to left:***

  `RL: ! prefix ++ -- casts * dereference & address sizeof`

  - Negation
  - Prefix increment/decrement
  - Casts
  - $*p$ (the object stored at location p)
  - & address
  - `sizeof()`

- `LR: * / %`

  $*/\%$ multiplication, division, remainder modulo
  Left to right.

- `LR: + -`

  $+-$ addition, subtraction
  Left to right.

- `LR: < <= > >=`

  $<, <=, >=, >$ relations
  Left to right.

- `LR: == !=`

  $==, !=$ relations
  Left to right.

- `LR:  &&`

  && logical AND
  Left to right.

- `LR:  ||`

  || logical OR
  Left to right.

- `RL:  =  +=  -=  *=  ........`

  $=$, $+=$, $-=$, etcetera Assignment and assignment operators
  Right to left.

Examples.

Disambiguate the following expressions by inserting parentheses, and say whether the expression is meaningful (legal), assuming the variables have suitable types.

```
  (i) while (   *x++!='\0'   )..
 (ii) a = b = c == 0
(iii) a = b == c = 0
 (iv) a = b = c == d && e || f || g
  (v)  * x[3] -> y[4]

  (i) while (   *x++!='\0'   )..
       while (  ( *(x++) ) != '\0' )..
legal
 (ii) a = b = c == 0
       a = ( b = ( c == 0 ))
legal
(iii) a = b == c = 0
       a = ( ( b == c ) = 0 )
illegal
 (iv) a = b = c == d && e || f || g
      a = ( b = ((( ( c == d ) && e  )|| f )|| g ) )
legal
  (v)  * x[3] -> y[4]
       * ( (x[3]) -> (y[4]) )
illegal
```