# 3.8 Order of evaluation

## **Precedence rules**

The order in which operators are evaluated in an expression are known as **precedence rules**. Arithmetic, logical, and relational operators are evaluated in the order shown below.

Table 3.8.1: Precedence rules for arithmetic, logical, and relational operators.

Operator/Convention	Description	Explanation
()	Items within parentheses are evaluated first	In (a * (b + c)) - d, the + is evaluated first, then *, then
!	! (logical NOT) is next	! x    y is evaluated as (!x)    y
*/%+-	Arithmetic operators (using their precedence rules; see earlier section)	z - 45 * y < 53 evaluates * first, then -, then <.
< <= > >=	Relational operators	<pre>x &lt; 2    x &gt;= 10 is evaluated as (x &lt; 2)    (x &gt;= 10) because &lt; and &gt;= have precedence over   .</pre>
== !=	Equality and inequality operators	<pre>x == 0 &amp;&amp; x &gt;= 10 is evaluated as (x == 0) &amp;&amp; (x &gt;= 10) because &lt; and &gt;= have precedence over &amp;&amp;. == and != have the same precedence and are evaluated left to right.</pre>
&&	Logical AND	x == 5    y == 10 && z != 10 is evaluated as

		(x == 5)    ((y == 10) && (z != 10)) because && has precedence over   .
II	Logical OR	has the lowest precedence of the listed arithmetic, logical, and relational operators.

Feedback?

PARTICIPATION ACTIVITY

3.8.1: Applying the precedence rules to an expression can be thought of as a 'tree'.



• 1

2

4



2x speed

$$x + 1 > y * z || z == 3$$

Next comes +, then >, then ==, and finally ||.

Feedback?

PARTICIPATION ACTIVITY

3.8.2: Order of evaluation.



To teach precedence rules, these questions intentionally omit parentheses; good style would use parentheses to make order of evaluation explicit.

- Which operator is evaluated first?
   y && x
  - y && x





2) Which operator is evaluated first? w + 3 > x - y \* z





! has one of the highest precedences, higher than &&, so !y is evaluated first. If y is false and x is true, the expression evaluates as (!false) && true, which is true && true, which is true.

#### **Correct**



Arithmetic operators (+, -, \*) have precedence over relational operators. Among +, -, \*, the \* has higher

precedence and thus y \* z is evaluated first.





3) In what order are the operators evaluated? w + 3 = y - 1 & x

- 4) To what does this expression evaluate, given int x = 4, int y = 7. x == 3 || x + 1 > y
  - O true
  - false

#### Correct

Arithmetic operators have higher precedence than equality or logical AND operators. Among + and -, evaluation is left-to-right, so + is first, then - is second. Next, the equality operators have precedence over logical AND, so != is third. && is thus fourth.

#### Correct

The arithmetic operator has highest precedence, yielding  $x == 3 \parallel 5 > y$ . Next is the relational operator >, yielding x  $== 3 \parallel \text{false}$  (because 5 > 7 is false). Next is ==, yielding false || false. Finally, the || is evaluated, yielding false.

Feedback?

## **Common error: Missing parentheses**

A <u>common error</u> is to write an expression that is evaluated in a different order than expected. <u>Good practice</u> is to use parentheses in expressions to make the intended order of evaluation explicit. Several examples are below.

#### **PARTICIPATION ACTIVITY**

3.8.3: Common errors in expressions.



1) Does! x == 3 evaluate as !(x == 3)?





2) Does w + x == y + zevaluate as (w + x) ==(y + z)?



() No

### Correct

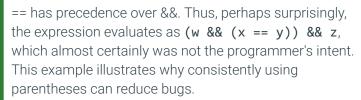
! has precedence over ==, so the expression evaluates as (!x) == 3. Using parentheses ensures an expression evaluates as a programmer desires, as in !(x == 3)

#### Correct

+ has precedence over ==, so w + x and y + x are each evaluated before the ==. However, good practice is to write (w + x) == (y + z) to make the intended order explicit.

- 3) Does w && x == y && z evaluate as (w && x) == (y && z)?
  - O Yes
  - No

#### **Correct**



Feedback?

## PARTICIPATION ACTIVITY

3.8.4: Order of evaluation.



Which illustrates the actual order of evaluation via parentheses?

- 1) ! green == red
  - (!green) == red
  - O!(green == red)
  - O (!green =)= red
- 2) bats < birds ||
  birds < insects</pre>
  - O ((bats < birds) || birds) < insects
  - O bats < (birds || birds) < insects
  - (bats < birds) ||
    (birds < insects)</pre>
- 3) ! (bats < birds) ||
   (birds < insects)</pre>
  - O!((bats < birds) || (birds < insects))
  - (! (bats < birds)) ||
     (birds < insects)</pre>
  - O ((!bats) < birds) || (birds < insects)
- 4) (num1 == 9) || (num2 == 0) && (num3 == 0)

#### Correct

Spacing doesn't matter; ! has precedence. Lack of parentheses causes common errors.

#### **Correct**

The comparisons occur first. The parentheses make that clear.

#### **Correct**

The items within parentheses are evaluated first. Then, ! has precedence over || so is evaluated next.

Correct

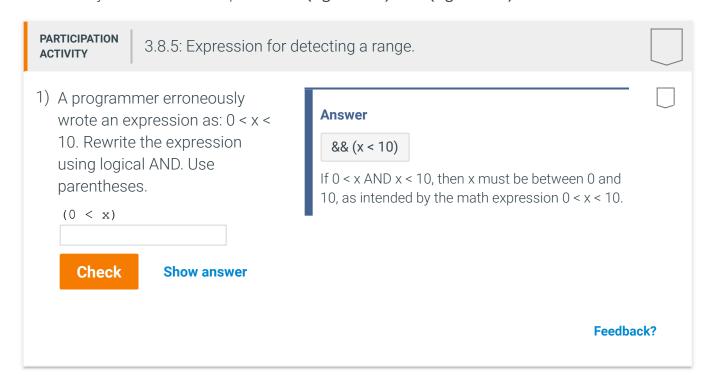
3.8. Order of evaluation (num1 == 9) || Items within parentheses are evaluated first. && has precedence over ||. ((num2 == 0) &&(num3 == 0)) $\bigcirc$  ((num1 == 9) || (num2 == 0)) &&(num3 == 0) $\bigcirc$  (num1 == 9) || (num2 == (0 &&num3) == 0Feedback?

## Common error: Math expression for range

A <u>common error</u> often made by new programmers is to write expressions like (16 < age < 25), as one might see in mathematics.

The meaning, however, almost certainly is not what the programmer intended. Suppose age is presently 28. The expression is evaluated left-to-right, so evaluation of 16 < age yields true. Next, the expression true < 25 is evaluated; clearly not the programmer's intent. However, true is actually 1, and evaluating 1 < 25 will yield true. Thus, the above expression evaluates to true, even for ages greater than 25.

Thus, 16 < age < 25 is actually the same as (16 < age) < 25, which evaluates to (true) < 25 for any age over 16, which is the same as (1) < 25, which evaluates to true. The correct way to do such a comparison is: (age > 16) && (age < 25).



## **Common error: Bitwise rather than logical operators**

Logical AND is && and not just &, and logical OR is || and not just |. & and | represent **bitwise operators**, which perform AND or OR on corresponding individual bits of the operands.

A <u>common error</u> is to use a bitwise operator instead of a logical operator, typing & instead of &&, or typing | instead of ||. A bitwise operator may yield different behavior than expected.

