EXPRESSIONS

Introduction to expressions

In computer programming an EXPRESSION can be either mathematical or logical. All expressions are EVALUATED.

Evaluating a MATHEMATICAL EXPRESSION generally follows the same rules you learned in math, but the result is usually assigned to a variable.

When evaluating a BOOLEAN EXPRESSION the result is always either “True” or “False”. This is usually the result of comparing a combination of constants (such as simple numbers), the values of variables, and also mathematical expressions, using a relational operator such as greater or less-than.

Greater-than and less-than are both relational operators because they compare to values. Other relational operators include “greater-than or equal to” (written >=), less-than or equal to (written <=), equal to (written ==) (YES, two equals signs!), and not equal to (written !=).

Using mathematical expressions

Mathematical expressions are frequently used when assigning a new value to a variable.

For example:

```
var x = 3 * 5 + 2
```

Uses the mathematical expression 3 times 5 plus 2 to assign the value 17 to the variable `x`.

Mathematical expressions can also include variables of course (remember, x now has the value 17):

```
var y = ( x – 1 ) / 2
```

In order to evaluate this mathematical expression, we substitute the current value of `x` so we get:

```
var y = ( 17 – 1 ) / 2
```

So the variable `y` gets the value 8.

Using Boolean expressions

Boolean expressions can also be assigned to variables, but in that case they will be either true or false:

```
var b = 5 < 10
```

The variable `b` now has the value `true` because 5 is less than 10.

Just like mathematical expressions, Boolean expressions can contain variables:

```
var c = x < y
```

Using the current values of `x` and `y`:

```
var c = 17 < 8
```

So `c` is `false` because 17 not less than 8!

The Logical or Conditional Operators

We are all familiar with mathematical operators such as + and -, but there is also a class of operators known as the logical operators. Logical operators are only used to compare Boolean values. There are really only two logical operators AND and OR.

In JAVA, the "AND" operator is written `&&` and the "OR" operator is written `||`.

The “AND” Operator: &&

The “AND” operator evaluates as “`true`” if both sides of the operator are “`true`,” thus `(1 < 2) && (2 == 2)` is “`true`” because `(1 < 2)` is “`true`” and `(2 == 2)` is “`true`.”

The “OR” Operator: ||

The “OR” operator evaluates as “`true`” if either sides of the operator are “`true`,” thus `(10 < 0) || (5 > 4)` is "`true`" because `(10 < 0)` is “`false`” but `(5 > 4)` is “`true`.”

Prior Variable Values

Variable	Value
x	17
y	8
b	true
c	false

Using the “AND” and “OR” Operators:

The logical operators can be used in variable assignments as part of a Boolean expression (using our variables values from before):

```
var d = b && c
```

Substituting our current variable values, the expression is evaluated as:

```
var d = true && false
```

For the result of an `&&` operator to be `true`, both sides must be `true`, and in this case, only one side is `true`, so `d` is `false`.

Now let’s look at a logical expression using the OR operator:

```
var e = b || c
```

Substituting we get:

```
var e = true || false
```

For the result of an `||` operator to be `true`, either side can be `true`, and in this case, at least one side is `true`, so `e` is `true`.

Compound Expressions:

A compound expression simply combines multiple expressions using the order of precedence. When combining mathematical and logical (or conditional) expressions, always evaluate the mathematical expressions first!

Analyze the expression when `i=4` and `j=2`.

Original Expression:	(	i * j > i + j	&&	j - i > 0	)	||	(	j < 0	)			
Substitute values:	(	4 * 2 > 4 + 2	&&	2 - 4 > 0	)	||	(	2 < 0	)			
Evaluate mathematical operators:	(	8 > 6	&&	-2 > 0	)	||	(	2 < 0	)			
Evaluate Inner-Boolean operators:	(	True	&&	False	)	||	(	False	)			
Evaluate outer-Boolean operator:	(	False			)	||	(	False	)	= False		

Try analyzing the expression when `i=2` and `j=-2`.

Original Expression:	(	i * j > i + j	&&	j - i > 0	)	||	(	j < 0	)			
Substitute values:	(	* > +	&&	- > 0	)	||	(	< 0	)			
Evaluate mathematical operators:	(	>	&&	> 0	)	||	(	< 0	)			
Evaluate Inner-Boolean operators:	(		&&		)	||	(		)			
Evaluate outer-Boolean operator:	(				)	||	(		)	=		

Now analyze the expression when `i=2` and `j=-5`.

Original Expression:	(	i * j > i + j	&&	j - i > 0	)	||	(	j < 0	)			
Substitute values:	(	* > +	&&	- > 0	)	||	(	< 0	)			
Evaluate mathematical operators:	(	>	&&	> 0	)	||	(	< 0	)			
Evaluate Inner-Boolean operators:	(		&&		)	||	(		)			
Evaluate outer-Boolean operator:	(				)	||	(		)	=		