# Expression Evaluation

**When operators and operands are evaluated, each operator is applied to its** operands, and a **temporary value** is calculated. This is the **result** of the expression.

Let's see how this expression is **evaluated**:

```
int a = 3 + 7 * 5 / 2 - 4;
```

1. There are **six operands**: the variable `a`s and five `int` literals, along with **five operators**: the assignment operator, multiplication, division, addition and subtraction.
2. Multiplication and division have higher precedence than addition or subtraction. However, the `*` and `/` operators are **tied** when it comes to dealing with the `5`. That means we have to fall back on **associativity**, going left-to-right, performing the multiplication before the division.

   ```
   a = 3 + ((7 * 5) / 2) - 4;
   ```

   Using parentheses we can represent the expression at this stage like this. Evaluating those subexpressions, we end up with:

   ```
   a = 3 + ((35) / 2) - 4;    // multiplication
   a = 3 + 17 - 4;            // division
   ```

3. Now we have three operands and two operators at the same precedence. Again, we fall back on associativity (left to right) and evaluate addition (on the left) and then subtraction (on the right).

   ```
   a = (3 + 17) - 4;    // addition
   a = 20 - 4;          // subtraction
   ```

4. The assignment operator has the lowest precedence of all, so we finish up by copying `16` into the variable `a` (this is the **side effect of the expression) and returning `16` as its value.**

In C and C++, the **order of operation** (specified by precedence and associativity) and the **order of evaluation** are not identical. Here's a simple example:

```
x = a() * b() + c();
```

Order of operation guarantees that the results of (`a() * b()`) will be calculated before the addition of `c()`. However **no guarantees** are made about the order in which the functions will be called: `c()` could be called first, or `a()` could be called first.

If functions have no side effects (**idempotent functions**) this doesn't make a difference. If functions have side effects, such as printing, the result is **undefined**.