Arguments evaluation order should not be relied on

Bug
Critical

- cppcoreguidelines
- unpredictable
- lock-in

Arguments evaluation order in a function call is not specified:

- Before C++17, the evaluation of each argument was unsequenced with the evaluation of other arguments, which can lead to undefined behavior if the same value is modified in several arguments,
- After C++17, it is sequenced, but in an unspecified order: the behavior is not longer undefined, but the values passed to the function will be non portable.

Both cases should be avoided, because the code will probably not be what was expected.

Noncompliant Code Example

```
void f(int i, int j);

void g() {
  int i = 0;
  f(++i, ++i); // Noncompliant, the call could either be f(1,2) or f(2,1)
(since C++17) or undefined behavior (before C++17)
}
```

Exceptions

This rule does not apply to overloaded operators because they respect the sequencing order rules of the operator they overload (since C++17).

See

- C++ Core Guidelines ES.44 - Don’t depend on order of evaluation of function arguments
- cppreference.com - order of evaluation