rules.sonarsource.com

C++ static code analysis: A single statement should not have more than one resource allocation

2-3 minutes

In a statement, the order of evaluation of sub-expressions (e.g. the arguments of a function call) is not totally specified. This means the compiler can even interleave the evaluation of these sub-expressions, especially for optimization purposes.

If you have several resource allocations in one statement, and the first succeeds while the second fails and throws an exception, the first allocated resource can leak. The classical mitigation for this issue is to use a RAII manager to wrap the raw resource, but since the execution order is not specified, you may not have the time to do this.

It is possible to write code that contains several allocations and will still behaves correctly (it has even been made easier in C++17, where the evaluation order has been made more strict), however it requires expert-level knowledge of the language. It is simpler, and more future-proof, to simply avoid using several allocations in a single statement.

Noncompliant Code Example

```
#include <memory>

using namespace std;

class S {
public:
  explicit S(int a, int b);
};

void g(shared_ptr<S> p1, shared_ptr<S> p2);

void f() {
  g(shared_ptr<S>(new S(1, 2)),
```

shared_ptr<S>(new S(3, 4))); // Noncompliant: 2 resources are allocated in the same expression statement

}

In this example, it would be valid for a pre-C++17 compiler to run the code in this order:

- `new S(1, 2) => p1`
- `new S(3, 4) => p2`
- `shared_ptr<S>(p1) => s1`
- `shared_ptr<S>(p2) => s2`
- `g(s1, s2)`

In that case, if the second allocation fails, the memory allocated for the first one will be leaked, since the `shared_ptr` has not yet been able to claim ownership of the object.

Compliant Solution

#include <memory>

using namespace std;

class S {

```
public:
  explicit S(int a, int b);
};

void g(shared_ptr<S> p1, shared_ptr<S> p2);

void f() {
  g(make_shared<S>(1, 2), make_shared<S>(3, 4)); // Compliant: no resource in the same expression statement
}
```

See

- C++ Core Guidelines R.13 - Perform at most one explicit resource allocation in a single expression statement
- cppreference - Order of evaluation