C++ static code analysis: "try_emplace" should be used with "std::map" and "std::unordered_map"

2-3 minutes

`emplace` and `insert` in `std::map` and `std::unordered_map` might construct the (key, value) pair, including the value object, even when it is not necessary.

`emplace` destroys the constructed pair if the key is already present, wasting the effort on construction and destruction of the value.

If `insert` was called with a temporary, it leads to an extra copy or move construction and destruction of the temporary.

C++17 introduced `try_emplace` that does not construct the value if the key is already present in the map and constructs the value in place if necessary.

In most cases, you should use `try_emplace`. In particular, if two conditions hold:

- You are inserting a single object at a time.
- You are creating a new mapped-to value and/or (key, value) pair just to insert it into the map.

You should keep the `insert` if one of the conditions holds:

- The (key, value) pair is already constructed (for another purpose).
- You want to insert multiple (key, value) pairs with a single call.

You should keep `emplace` and `emplace_hint` if

- You use piecewise construction with `std::piecewise_construct`.

This rule detects calls to `insert` that lead to the construction of a large temporary object, as well as calls to `emplace` and `emplace_hint` with no piecewise construction.

Noncompliant Code Example

```
void f() {
  std::map<int, std::string> bodies({{3, "Lorem ipsum..."}});
  bodies.emplace(3, "Lorem ipsum..."); // Noncompliant
  bodies.insert({3, "Lorem ipsum..."}); // Noncompliant
}
```

Compliant Solution

```
void f() {
  std::map<int, std::string> bodies({{3, "Lorem ipsum..."}});
  bodies.try_emplace(3, "Lorem ipsum..."); // Compliant
  auto p = std::make_pair(3, "Lorem ipsum..."); // The (key, value)
pair is already constructed for another purpose
  bodies.insert(p); // Compliant
  use_the_pair(p);
}
```

Exceptions

You should keep `insert` for exception safety if your mapped-to type is a smart pointer and the argument is a `new` expression.