rules.sonarsource.com

C++ static code analysis: "std::optional" member function "value_or" should be used

2 minutes

C++17 introduces `std::optional<T>`, a template class which manages an optional contained value. By default, the container doesn't contain any value. The contained value can be accessed through member functions like `value()`, `operator*()`, or `operator->()`. Before accessing the value it is a good practice to check its presence using `has_value()` or `operator bool()`.

`value_or(default_value)` member function returns the contained value if present or `default_value` otherwise. This rule flags patterns which could be simplified by a single call to `value_or(default_value)` instead of two steps logic:

- check presence, i.e. with `has_value`
- use `value()` if present, `default_value` otherwise

Noncompliant Code Example

```
void fun(const std::optional<std::string> &arg) {
  if (arg.has_value()) { // Noncompliant, the entire if statement can be simplified to a simpler statement using "value_or(default_value)"
    std::cout << arg.value();
  } else {
    std::cout << "magic";
  }

  // another way to check presence and access value
  std::cout << (arg ? *arg : "magic"); // Noncompliant, replace the ternary operator by using "value_or(default_value)"
}

void moveFun(std::optional<std::string> arg) {
  if (arg.has_value()) { // Noncompliant, the entire if statement can be simplified to a simpler statement using "value_or(default_value)"
    sink(std::move(arg.value()));
  } else {
    sink("magic");
  }
}
```

Compliant Solution

```
void fun(const std::optional<std::string> &arg) {
  std::cout << arg.value_or("magic"); // Compliant, neat and simple
}

void moveFun(std::optional<std::string> arg) {
  sink(std::move(arg).value_or("magic"));
}
```