rules.sonarsource.com

C++ static code analysis: Elements in a container should be erased with "std::erase" or "std::erase_if"

2 minutes

Removing elements with a specific value or that follow a given predicate from a container is a common task. Before C++20, this was not straightforward. The way to do it had to depend on the type of your container.

For sequence containers, you would end up following what is called the *erase-remove idiom*:

- Call `std::remove` or `std::remove_if` with, as parameters, the container and the criterion to fulfill
- Call the container member function `erase` on the result

For associative containers, you would have no other option than looping through all the elements by hand.

However, C++20 introduced two new methods: `std::erase` (for sequence containers only) and `std::erase_if` which erase all elements equal to a value or that satisfy a given predicate.

This rule raises an issue when `std::erase` or `std::erase_if` could be used to simplify the code.

Noncompliant Code Example

```
void removeZeros(std::vector<int> &v) {
  v.erase(std::remove(v.begin(), v.end(), 0), v.end()); // Noncompliant
}

void removeOddNumbers(std::vector<int> &v) {
  v.erase(std::remove_if(v.begin(), v.end(), [](auto i) { return i%2 ==
0; }), v.end()); // Noncompliant
}

void removeOddNumbers(std::unordered_map<std::string, int> &m)
{
  auto it = m.begin();
  while (it != m.end()) { // Noncompliant
    if (it->second % 2 == 0) {
      it = m.erase(it);
    } else {
      ++it;
    }
  }
}
```

Compliant Solution

```
void removeZeros(std::vector<int> &v) {
  std::erase(v, 0);
}

void removeOddNumbers(std::vector<int> &v) {
  std::erase_if(v, [](auto i) { return i%2 == 0; });
}

void removeOddNumbers(std::unordered_map<std::string, int> &m)
{
  std::erase_if(m, [](auto item) { return item.second % 2 == 0; });
}
```