C++ static code analysis: Result of the standard remove algorithms should not be ignored

2-3 minutes

Despite their names, the standard remove algorithms (`std::remove`, `std::remove_if`, `std::unique`) do not remove the elements. Instead, they rearrange elements so that the preserved (not removed) elements are stored first and return an iterator to the end of that sub-sequence (the content of the remainder of the container is unspecified). The responsibility for removing the remaining elements lies in the hands of the programmer, leading to what is called the *erase-remove* idiom.

Ignoring the return of one of these functions is an indication of a bug, as further operations on the container may access elements with unspecified values.

The `range` counterparts of the above algorithms (in namespace `std::ranges`) return a `subrange` of remaining elements, but otherwise, they exhibit the same behavior.

The rule raises an issue when the result of the invocation

of `remove`, `remove_if` or `unique` from namespace `std` or `std::ranges` is ignored.

Noncompliant Code Example

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

auto largestFiles(vector<char*> fileNames, int n) {
  remove(fileNames.begin(), fileNames.end(), nullptr); // Noncompliant
  return max_element(fileNames.begin(), fileNames.end(),
    [](auto f1, auto f2) {
      // Undefined behavior: accessing unspecifed elements and potentially dereferencing null pointers
      return filesystem::file_size(f1) < filesystem::file_size(f2);});
}
```

Compliant Solution

```
void func(std::vector<int>& v) {
  std::erase_if(v, [](int x) { return x % 2 == 0; });
}
// Or, before C++20:
void func(std::vector<int>& v) {
  v.erase(std::remove_if(v.begin(), v.end(), [](int x) { return x % 2 == 0; }), v.end());
```

```
}

auto largestFiles(vector<char*> fileNames, int n) {
  // Compliant, the value is used (without the erase-remove
idiom, it did not apply here)
  auto it = remove(fileNames.begin(), fileNames.end(),
nullptr);
  return max_element(fileNames.begin(), it,
    [](auto f1, auto f2) {
      return filesystem::file_size(f1) <
filesystem::file_size(f2);});
}
```

See

- {rule:cpp:S6165} - replacing erase-remove idiom with `std::erase`/`std::erase_if`