C++ static code analysis: Capture by reference in lambdas used locally

2 minutes

If the lifetime of the arguments passed to a lambda function is longer than the lifetime of the lambda itself, these arguments can be passed by reference.

Doing so avoids copying potentially big objects and it should be preferred over using copy capture.

This rule reports an issue if a lambda passed into an algorithm that uses it locally (all algorithms in headers `<algorithm>` and `<numeric>`) captures some values.

Noncompliant Code Example

```
using Vec = vector<int>;

Vec applyPermutation(const Vec& v, const Vec& permutation) {
  assert(v.size() == permutation.size());

  const auto n = v.size();
  Vec result(n);

  // Noncompliant: this will copy the entire v vector for each
iteration, resulting in n^2 operations
  transform(permutation.begin(), permutation.end(),
back_inserter(result),
    [v](int position){ return v[position]; });

  return result;
}
```

Compliant Solution

```
using Vec = vector<int>;

Vec applyPermutation(const Vec& v, const Vec& permutation) {
  assert(v.size() == permutation.size());

  const auto n = v.size();
  Vec result(n);

  // Compliant: this will NOT copy the entire v vector for each
iteration, resulting in n operations
  transform(permutation.begin(), permutation.end(),
back_inserter(result),
    [&v](int position){ return v[position]; });

  return result;
}
```

See

- C++ Core Guidelines F.52 - Prefer capturing by reference in lambdas that will be used locally, including passed to algorithms