Wording for [[maybe_unused]] attribute.

Document No.: P0212R0
Project: Programming Language C++ - Evolution
Author: Andrew Tomazos <andrewtomazos@gmail.com>
Date: 2015-01-31

Summary

A wording for the [[maybe_unused]] attribute is proposed for application to the C++17 Working Draft. The semantics of [[maybe_unused]] are the same as those described for [[unused]] in P0068R0 and presented to Kona EWG. Kona EWG voted SF=5, F=11, N=2, A=0, SA=0 in favor of the attribute.

See P0068R0 for detailed motivation/rationale.

Changes From P0068R0

- The Kona EWG bikeshedding vote resulted in EWG guidance to change the name from [[unused]] to [[maybe_unused]]. The wording includes this change.
- There was a feature that interacted with [[nodiscard]] that EWG guided was not desired. This has been removed from the proposal (see P0189R0 for details).

Wording

7.6.6 Maybe unused attribute [dcl.attr.unused]

1. The attribute-token **`maybe_unused`** can be used to mark various names and entities that may be intentionally not used. [Note: If an implementation would have otherwise emitted a warning about an entity, so marked, not being used, they are encouraged not to. -- end note] [Note: Implementations are discouraged from emitting a warning if an entity marked **`maybe_unused`**, is used. -- end note] It shall appear at most once in each attribute-list, with no attribute-argument-clause.
2. The attribute may be applied to the declaration of a class, a typedef-name, a variable, a non-static data member, a function, an enumeration or a template specialization.
3. A name or entity declared without the **`maybe_unused`** attribute can later be re-declared with the attribute and visa-versa. An entity is considered marked after its first declaration

that is marked is analyzed, and for the remainder of translation of the current translation unit.

Examples

Example 1

Compiled with an unused variables warning enabled in a release build (NDEBUG):

```
std::pair<int, int> plot_to_curve(int x, int z) {
  int y = save_and_project(x,z); // WARNING: y unused
  assert(y == 0);
  return {x, z};
}

std::pair<int, int> plot_to_curve(int x, int z) {
  [[maybe_unused]] int y = save_and_project(x,z); // OK
  assert(y == 0);
  return {x, z};
}
```

Example 2

When compiled with USE_IMPL1 defined:

```
static void impl1() { ... }
static void impl2() { ... } // warning: impl2 unused

void iface() {
#ifdef USE_IMPL1
  impl1();
#elif USE_IMPL2
  impl2();
#else
  #error set an implementation
#endif
}

[[maybe_unused]] static void impl1() { ... }
[[maybe_unused]] static void impl2() { ... } // OK

void iface() {
#ifdef USE_IMPL1
  impl1(); // OK
#elif USE_IMPL2
  impl2(); // OK
#else
  #error set an implementation
#endif
}
```

FAQ

1. What constitutes an entity being used?

As per the existing appearance of the term “used” in [dcl.attr.deprecated], this is unspecified and hence left as a quality of implementation issue. There are a spectrum of increasingly complex algorithms an implementation could use to statically analyze a little-used entity in a given program, and to take an educated guess as to whether it likely enough to be indicative of an logic error to issue a warning. We feel it would be onerous and unnecessarily restrictive on implementations to strictly specify a particular algorithm.

2. Why do you discourage implementations from emitting a warning if an [[maybe_unused]] entity is used?

This is in line with existing practice. It shows that many times the [[maybe_unused]] annotation is used where a certain set of defines leads to an entity appearing unused for one preprocessed translation unit, as typified by the assert-NDEBUG case. If the implementation warned when an [[maybe_unused]]-marked entity was used, this would trigger warnings when the other set of defines were used:

ie If an implementation warned about an [[maybe_unused]]-marked entity being used, then without NDEBUG the following would generate a warning:

```
std::pair<int, int> plot_to_curve(int x, int z) {
  [[maybe_unused]] int y = save_and_project(x,z);
  assert(y == 0); // WARNING: y used ???
  return {x, z};
}
```

This would of course defeat the purpose. The semantic of [[maybe_unused]] is that the entity MAY appear unused, not that it MUST be unused.

It is prohibitively difficult for an implementation to statically analyze a translation unit under all possible results of preprocessing.