The maybe_unused attribute

Reply-to: Aaron Ballman (aaron@aaronballman.com; aballman@cert.org)

Document No: N2053
Date: 2016-04-12

Introduction

Unused variables in source code are often an indirect indication that something may be wrong with the user's source code. Sometimes it's simply a vestigial variable resulting in benign reduced code quality. Sometimes it's an accidental omission that can lead to unexpected program behavior. Because of this, many implementations issue a diagnostic warning the user when an entity is unused. However, under some circumstances, this disuse is because of macros, so the entity is only unused under certain configurations.

Proposal

This document proposes the `[[maybe_unused]]` attribute as a way for a programmer to specify that an entity may appear to be unused for a particular program configuration, but it is an expected outcome the programmer is aware of. This allows programmers to specify their intent explicitly, giving an implementation the opportunity to reduce the number of false positive diagnostics it emits and reduces the risk of programmers writing "clever" code to silence those false positives, which would otherwise introduce possible negative effects on program correctness or performance.

Consider:

```
struct point {
    int x, y;
  };

  struct point plot_to_curve(int x, int z) {
    int y = project_point(x, z); // (1)
    assert(y == 0 && "Unexpected projected value"); // (2)

    struct point ret = {x, z};
    return ret;
  }
```

The variable y is declared and initialized at (1), and is used by the assert macro at (2). However, when the `NDEBUG` macro is defined, the `assert` macro is defined simply as `#define assert(ignore) ((void)0)`, which results in the variable y being diagnosed as unused. Instead, the `[[maybe_unused]]` attribute can be specified to explicitly state programmer intent more clearly, e.g., `[[maybe_unused]] int y = project_point(x, z);`

The `[[maybe_unused]]` attribute can be applied to the declaration of a `struct`, `enum`, `union`, `typedef`, variable (including member variables), function, or enumerator. Implementations are encouraged to not emit a diagnostic when such an entity is unused or when the entity is used despite being marked as `[[maybe_unused]]`. What constitutes a "use" is left to QoI due to the fact that there are a considerable number of heuristics that can be implemented to take an educated guess as to whether a particular code pattern qualifies as a use or not.

Rationale

The `[[maybe_unused]]` attribute has real-world use, being implemented by Clang and GCC as vendor-specific extensions under the name `__attribute__((unused))`, and was standardized under the name `[[maybe_unused]]` by WG21. Other common idioms used to express the intent are:

```
__pragma(warning(suppress:4100))
#pragma foo diagnostic unused-bar ignored
void f(int /*arg*/);
#define UNUSED(x) (void)(x)
#ifdef COMPILER1
  #define UNUSED something
#elif COMPILER2
  #define UNUSED something
#endif
int v = initializer; v = v;
```

Configurations where an entity may or may not be used, while somewhat uncommon, do arise in well-written code such as with use of the `assert()` macro. This attribute provides users with a uniform way to express intent to the implementation without sacrificing code quality or readability.

Proposed Wording

This proposed wording currently uses placeholder terms of art and is expected to change as N2049 progresses. It assumes a new subclause, 6.7.11, Attributes that describes the referenced grammar terms. The [Note] in paragraph 1 of the semantics is intended to convey informative guidance rather than normative requirements.

6.7.11.3 Maybe_unused attribute

Constraints

1 The *attribute-token* `maybe_unused` indicates that a name or entity is possibly intentionally unused. It shall appear at most once in each *attribute-list* and no *attribute-argument-clause* shall be present.

2 The attribute shall be applied to the declaration of a struct, a union, a *typedef-name*, a variable, a non-static data member, a function, an enumeration, or an enumerator.

Semantics

1 [Note: For an entity marked `maybe_unused`, implementations are encouraged not to emit a warning that the entity is unused, or that the entity is used despite the presence of the attribute. -- end note]

2 A name or entity declared without the `maybe_unused` attribute can later be redeclared with the attribute and vice versa. An entity is considered marked after the first declaration that marks it.

3 EXAMPLE

```
[[maybe_unused]] void f([[maybe_unused]] int i) {
  [[maybe_unused]] int j = i + 100;
```

```
    assert(j);
  }
```

Implementations are encouraged not to warn that `j` is unused, whether or not `NDEBUG` is defined.

Acknowledgements

I would like to recognize the following people for their help in this work: David Keaton, David Svoboda, and Andrew Tomazos. I would also like to thank the US Department of Homeland Security, without whose funding this proposal would not have been made.

References

[N2049]
Attributes in C. Aaron Ballman. http://www.open-std.org/jtc1/sc22/wg14/www/docs/n2049.pdf

[P0068R0]
Proposal of [[unused]], [[nodiscard]] and [[fallthrough]] attributes. Andrew Tomazos. http://www.open-std.org/jtc1/sc22/wg21/docs/papers/2015/p0068r0.pdf

[P0212R1]
Wording for [[maybe_unused]] attribute. Andrew Tomazos. http://www.open-std.org/jtc1/sc22/wg21/docs/papers/2016/p0212r1.pdf