The fallthrough attribute

Reply-to: Aaron Ballman (aaron@aaronballman.com; aballman@cert.org)
Document No: N2052
Date: 2016-05-26

Introduction

A lesser-used idiom within a switch statement is the ability to "fallthrough" from a one case block into another case block. This can create expensive logic errors when the fallthrough behavior is unintentional because the secondary case functionality may appear to be reasonable at first glance, causing the incorrect behavior to be discovered elsewhere in the code base. Compilers cannot easily diagnose fallthrough behavior because there is no way to distinguish programmer intent: sometimes fallthrough behavior is expected and desirable, other times it is a bug.

Proposal

This document proposes the `[[fallthrough]]` attribute as a way for a programmer to specify that fallthrough behavior is desirable under the assumption that silently fallthrough is an accidental omission of a break statement. This allows programmers to specify their intent explicitly, giving an implementation the opportunity to diagnose fallthrough behavior in the absence of explicit marking.

Consider:

```
enum E { Zero, One, Two, Three };
 size_t calculate_required_buffer_size(enum E e) {
   size_t Ret = 0;
   switch (some_value) {
   case Zero: // (0)
   case One:
     Ret = 100;
     break;
   case Two:
     Ret = 200; // (1)
   case Three:
     Ret = 20; // (2)
     break;
   default:
     assert(0 && "Unknown enumeration value");
   }
   return Ret;
 }
```

If the enumerator `Two` is passed to `calculate_required_buffer_size()`, the `Ret` variable will store the value `200` at (1). However, because there is no break statement following the assignment, execution "falls through" to the next statement, overwriting `Ret` with the value `20` at (2). This results in a likely unexpected value being returned from `calculate_required_buffer_size()` that could, e.g., lead to the caller allocating an insufficient amount of memory for an object, resulting in an exploitable buffer overrun. Without an annotation, the compiler has insufficient information to

determine whether falling through from one case to another is intended behavior. Thus, the fallthrough into (2) can be diagnosed by a compiler. If the fallthrough is desired, the programmer can instead write:

```
// ...
case Two:
  Ret = 200;
  [[fallthrough]];
case Three:
  // ...
```

The fallthrough at (0) is left to QoI as to whether a diagnostic is desired or not. Common practice is for (0) to not diagnose fallthrough because there is only a single newline between the case labels.

The `[[fallthrough]]` attribute can only be applied to a null statement that precedes a case or default label statement within a switch statement.

Rationale

The `[[fallthrough]]` attribute has real-world use, being implemented by Clang and GCC as C++ vendor-specific attributes, and was standardized under the name `[[fallthrough]]` by WG21. This attribute cannot be implemented via the `__declspec` or `__attribute__` vendor-specific extensions because it is an attribute that appertains to a statement instead of a declaration or a type.

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.

6.7.11.2 Fallthrough attribute

Constraints

1 The *attribute-token* `fallthrough` shall be applied to a null statement (6.8.3); such a statement is a fallthrough statement. The *attribute-token* `fallthrough` shall appear at most once in each *attribute-list* and no *attribute-argument-clause* shall be present. A fallthrough statement may only appear within an enclosing `switch` statement (6.8.4.2). The next statement that would be executed after a fallthrough statement shall be a labeled statement whose label is a case label or default label for the same `switch` statement.

Semantics

1 [Note: The use of a fallthrough statement is intended to suppress a warning that an implementation might otherwise issue for a case or default label that is reachable from another case or default label along some path of execution. Implementations are encouraged to issue a warning if a fallthrough statement is not dynamically reachable. -- end note]

2 EXAMPLE

```
void f(int n) {
  void g(void), h(void), i(void);
  switch (n) {
```

```
    case 1:
    case 2:
      g();
      [[fallthrough]];
    case 3: // warning on fallthrough discouraged
      h();
    case 4: // fallthrough warning encouraged
      i();
      [[fallthrough]]; // constraint error
    }
  }
```

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

[P0188R1]
Wording for [[fallthrough]] attribute. Andrew Tomazos. http://www.open-std.org/jtc1/sc22/wg21/docs/papers/2016/p0188r1.pdf