Proposal for C2x
WG14 N2161

Title:	Harmonizing left-shift behavior with C++
Author, affiliation:	Aaron Ballman, GrammaTech
Date:	2017-07-15
Proposal category:	New features
Target audience:	Developers working on combined C and C++ code bases

Abstract: C++14 changed the behavior of the left-shift operator so that a common case of undefined behavior is instead well-defined behavior. Specifically, shifting a 1 into the sign bit of a signed operand to << is only undefined behavior on architectures other than two's complement.

Prior art: C++. GCC, Clang, and MSVC do not appear to treat this as an optimization opportunity (taking advantage of the undefined behavior), at least when targeting architectures that are two's complement and compiling for C.

Harmonizing left-shift behavior with C++

Reply-to: Aaron Ballman (aaron@aaronballman.com)
Document No: N2161
Date: 2017-07-15

Introduction

C++14 accepted a Defect Report regarding the treatment of shifting a one into the sign bit position of a signed integer value [CWG 1457]. The defect report noted that it was undefined behavior to shift a one into the sign bit, despite this being a common occurrence in the wild with reasonable, consistent results on two's complement architectures. This was discussed in C11 DR 463 and was determined to not be a defect, but was something to consider for the next revision of the standard [DR 463].

Rationale

While shifting into the sign bit of a signed integer is not considered to be a good practice, it does not seem prudent for it to result in unbounded undefined behavior when the operation has a reasonable meaning for the most common architecture found today. It is less user-hostile to allow such a construct to have well-defined meaning on a two's complement architecture, while remaining undefined behavior on other architectures. This improves code portability and security in the common case while retaining implementation freedom for architectures that cannot support the construct.

This functionality was requested in C++ because the undefined behavior produced by such a left-shift expression prevents the value from being used as an integer constant expression, which resulted in breaking a significant amount of code. C shares this concern due to the constraint violation in 6.6p4.

A search for "1 << 31" of source code on GitHub returns thousands of results in C and C++ source files [Example 1, Example 2]. While these are not examples of good coding practice, it does demonstrate usage in the wild.

Harmonizing C's treatment of this expression with C++'s will reduce user surprise with reasonable-looking constructs, especially constant expressions where the user elides a literal suffix. For instance, the following code is valid C++14 but invalid C code (and some implementations will diagnose this code in C mode but not in C++ mode).

```
#include <assert.h>
static_assert((1 << 31) < 0, "Oops");
```

When compiled in C11 mode, the preceding code produces a diagnostic with GCC 7.1 ("warning: expression in static assertion is not an integer constant expression") to conform to the constraint violation in 6.6p4 but does not produce a diagnostic in C++14 mode. Such a construct could easily be in a header file that is not under user control (such as a library header file), forcing the user to disable otherwise useful diagnostics if they choose to build while treating warnings as errors.

Beyond being an invalid constant expression, there are security concerns due to unbounded runtime undefined behavior with similar constructs. Consider:

```
void func(int shift) {
  int val = 1 << shift;

  if (shift >= (sizeof(int) * CHAR_BIT) - 1) {
    // Ensure we don't launch the missiles.
    return;
  }
  do_something_that_launches_missiles(val);
}
```

With a sufficiently clever optimizer, the code in the `if` statement might not be executed because the expression `1 << shift` informs the optimizer that shift amount must be less than 31 (assuming a 32-bit `int`), so the code can be optimized away as "dead" code. See CVE-2009-1897 for a similar example of this spooky action at a distance involving a null pointer dereference that resulted in a privilege escalation compromise of the Linux kernel [CVE].

Proposed Wording

The wording proposed is a diff from ISO/IEC 9899-2011. Green text is new text, while ~~red~~ text is deleted text.

Modify 6.5.7p4:

The result of `E1 << E2` is `E1` left-shifted `E2` bit positions; vacated bits are filled with zeros. If `E1` has an unsigned type, the value of the result is $\mathtt{E1} \times 2^{\mathtt{E2}}$, reduced modulo one more than the maximum value representable in the result type. If `E1` has a signed type and nonnegative value, and $\mathtt{E1} \times 2^{\mathtt{E2}}$ is representable in the corresponding unsigned type of the result type, then that is the resulting value, converted to the result type; otherwise, the behavior is undefined.

References

[CWG 1457]
C++ Standard Core Language Defect Reports and Accepted Issues, Revision 97. <unknown>. http://www.open-std.org/jtc1/sc22/wg21/docs/cwg_defects.html#1457

[CVE]
CVE-2009-1897. <US-CERT/NIST>. https://nvd.nist.gov/vuln/detail/CVE-2009-1897

[DR 463]
Defect Report Summary for C11 Version 1.11. <unknown>. http://www.open-std.org/jtc1/sc22/wg14/www/docs/n2109.htm#dr_463

[Example 1]
https://github.com/moyix/panda/blob/master/qemu/hw/mmc.h#L107

[Example 2]
https://github.com/llvm-mirror/llvm/blob/master/include/llvm/CodeGen/MachORelocation.h#L38