C and C++ Compatibility Study Group Meeting Minutes (Dec 2021)

Reply-to: Aaron Ballman (aaron@aaronballman.com)
Document No: N2883
SG Meeting Date: 2021-12-09

Thur Dec 09, 2021 at 1:00pm EST

Attendees

Aaron Ballman	WG21/WG14	chair
Philipp K. Krause	WG14	
Robert Seacord	WG14	
Martin Uecker	WG14	
Tom Honermann	WG21	scribe
Davis Herring	WG21	
Ville Voutilainen	WG21/WG14	
Hans Boehm	WG21	
JeanHeyd Meneide	WG21/WG14	chair
Nevin Liber	WG21	
Corentin Jabot	WG21	
Will Wray	(21)/(14)	

Code of Conduct: follows ISO, IEC, and WG21 CoCs (no current WG14-specific CoC)

Agenda

Discussing the following papers:

WG14 N2808 (http://www.open-std.org/jtc1/sc22/wg14/www/docs/n2808.htm) Allow 16-bit ptrdiff_t again
WG21 P1494R2 (https://wg21.link/p1494r2) Partial program correctness
WG21 P2460R0 (https://wg21.link/p2460r0) Relax requirements on wchar_t to match existing practices

WG14 N2808 Allow 16-bit ptrdiff_t again

Philipp provided an introduction

Davis: I don't think there is a requirement that ptrdiff_t be able to represent the distance from the start to the end of the object, though it is inconvenient if it is unable to.

Philipp: It isn't required in C either.

Ville: Browsed C++ draft and implementation quantity limits are different. There is a recommended minimum, but that isn't required. Ptrdiff_t does need to be able to represent the distance between two array subscripts. There does not seem to be an issue for C++.

Aaron: In C23 we added a bunch of _WIDTH macros. In C++, they are defined in terms of C headers. I think Ville is right though; this isn't a problem for C++.

Ville: You could run into the funny situation where the PTRDIFF_WIDTH macro disagrees with the actual width of an object.

Davis: In practice, that may not be an issue since C++ implementations may not target cases where this happens.

Philipp: gcc targets AVR where this could be an issue.

Davis: Such an implementation may just not be a fully conforming implementation.

JeanHeyd: Since PTRDIFF_WIDTH was introduced in C23, there isn't really any code in the wild that is affected by this, so this is unlikely to matter.

Aaron: Do you have the direction you need?

Philipp: Yes, there seem to be no C++ issues.

Aaron: Are there any library concerns that should be raised?

Davis: Would be polite to let them know.

Ville: We just inherit the macro, so implementations will know what to do.

Ville: Implementations won't actually have to change; AVR is 16-bit anyway.

P1494R2 Partial program correctness

Davis provided an introduction.

Martin: First comment: UB in C is not allowed to go back and chance observable behavior.

Martin: Second comment: We have a UB study group that should probably review this,.

Martin: Are there compilers that can go back and remove a printf()?

Aaron: There are cases of if statements being removed.

Davis: I did check for compilers that elide code.

Tom: Coverity does diagnose these time travel optimizations.

Martin: I would like to see an example where that happens.

Martin: Would prefer to rule out the possibility of that happening.

Davis: The situation in C does not appear to be very clear. The UB definition in C states that "possibilities may range from aborting with a diagnostic or ignoring the situation completely with unpredictable results".

Davis: I believe common compilers do not differentiate C and C++ in this regard.

Davis: We don't know how to specify UB such that time-travel scenarios are prohibited.

Martin: There is a C WG that is looking into a way to say that. The proposed std::observable() seems to be a way of stating such a requirement.

Davis: If that (inserting std::observable() calls) was done implicitly, that would prohibit optimizations.

Davis: If WG14 comes up with something that works, that would likely be preferred over this approach.

Philipp: One one hand, UB is UB and restricting it seems weird; limiting the powers of demons flying out of your nose.

Philipp: If you have a function with lots of local variables with addresses that are not taken, a call to any function would prohibit moving calculations across call boundaries if we implicitly required observable behavior.

Davis: Compilers are already prohibited from optimizing around function calls to functions that it can't see the source of because it can't determine that the program won't abort while in a call to such a function.

Davis: A call to std::observable() is equivalent to calling a function that does nothing but that the compiler is unable to see to determine that.

Philipp: That is an argument in favor of this proposal; this can be more performant than just adding an actual empty function.

Davis: In the examples in the paper, note that the calls to fprintf() are intentionally made to stderr because that requires the stream be flushed.

Hans: I posted an article in the chat that is relevant.

https://dl.acm.org/doi/pdf/10.1145/3466132.3468263

Hans: There is an issue with regard to whether all cases of UB have a clearly associated source location.

Davis: The possibility of data races was raised by SG1 as a case that does not have a clearly associated source location.

Davis: The wording associates time points with the check points.

Hans: I missed that SG1 discussion.

Davis: I'll try to find the meeting minutes.

Hans: I think there is an issue with the "happens before" wording in the proposal; I'll look more closely and follow up offline.

Davis: There are probably edge cases not fully accounted for here.

Ville: There are some implementation vendors that are concerned about performance here. This is no worse than a call to an opaque function call, but implementations would like to avoid having to spill registers as-if such a function were called.

Davis: It should be possible to treat this as an inlined function that performs some I/O.

Ville: This then requires an intrinsic with associated AST notation. We'd like to see some implementation experience that demonstrates a high quality implementation that avoids register spills; not just an implementation as an opaque function.

Davis: I'm unable to do that work, but look forward to hearing about such work.

JeanHeyd: It sounds like there is some concern that this may interfere with other work in WG14 being done regarding UB. However, this might also provide a tool that would make such work simpler.

JeanHeyd: We can take a poll to determine if we see any concerns with this approach.

Davis: Sounds like a fine idea.

Davis: WG14 may want to spend more time attempting to remove time-travel UB; I would like to hear back on such work regardless of how it works out.

JeanHeyd: I'll post this to the WG14 email list for additional comments.

The time-honored practice of deciding on wording for a possible poll commenced.

Poll: SG22 does not see any compatibility concerns for the direction of P1494r2.
Attendees: 11 (though Aaron had stepped away)
No objection to unanimous consent.

P2460R0 Relax requirements on wchar_t to match existing practices

Corentin provided an introduction.

Philipp: This makes wchar_t just another char with another name.

Philipp: Do we need wchar_t any more? Can we just make it UB?

Corentin: It would be user hostile to declare that code that exists today is now UB.

Corentin: I agree that wchar_t failed to do what it was intended to do and that we should not progress it further. But people do use it today. We would need a long term approach to replace it.

Corentin: What we could do is deprecate library functions that don't work.

Philipp: Microsoft is non-conforming here. UB is one way for implementations to have implementation-defined behavior. Is wchar_t still useful?

Philipp: Another approach would be to specify that wchar_t is either UTF-16 or UTF-32.

Corentin: That does not represent existing practice. On many POSIX systems, wchar_t will be UTF-32, but on FreeBSD and IBM systems, it may be something else; wide-EBCDIC for example.

Philipp: I have an implementation that had an 8-bit wchar_t.

Corentin: Tom had done some research and found that, on some DSP systems, that wide strings were still just ASCII.

Corentin: What to do with the library is a long term concern not addressed by this paper. This just proposes a limited change to better reflect existing behavior.

JeanHeyd: At the last WG14 meeting, someone suggested removing wchar_t support.

JeanHeyd: It was noted that it doesn't meet the requirements that were set out for it; everything about it is implementation-defined.

Tom: Microsoft did nothing wrong; they are a victim of progress.

Tom: wchar_t is useful and used, it just isn't useful in a portable program.

Tom: The future portable type is char8_t and UTF-8.

Corentin: The point that Microsoft did nothing wrong is a good point; that was the motivation for the paper. We should fix this situation for them.

JeanHeyd: IBM is also non-conforming here; On 32-bit AIX, they use UCS-2/UTF-16 or EUC-JP. On 64-bit systems, they have switched to UTF-32.

JeanHeyd: Recognizing that wchar_t is implementation-defined is probably a more useful way forward.

Corentin: JeanHeyd's proposal for new conversion functions would help both C and C++ by providing a library conversion feature that works with wchar_t.

JeanHeyd: Yes, I'm still working on that for C23.

Philipp: This won't cause problems for existing implementations, but an implementation could take advantage of this feature to change wchar_t to an 8-bit type and UTF-8 encoding.

Tom: That might be an improvement; an implementation could introduce a new ABI.

Philipp: In small device environments, ABI doesn't tend to matter much; all code gets recompiled.

Philipp: Users could see performance improvements, but might want to preserve C17 compatibility.

Tom: I think we can trust that implementors will not be hostile to their users.

Corentin: Many embedded compilers only support ASCII right now as is.

Philipp: There are also embedded compilers that support UTF-8.

Poll: SSG22 does not see any compatibility concerns for the direction of P2460r0 in C++.
Attendees: 10
No objection to unanimous consent

Aaron: The wording uses the term "code unit"; do we have a definition for that?

Tom: I thought there was a definition in either C or C++.

Aaron: There is a definition in C++, [lex.charset]p5. http://eel.is/c++draft/lex.charset#5

Wrapup

JeanHeyd: that concludes things.

Aaron: I'll send out a Doodle poll for meetings in January.

End at 2:34pm EST