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

Reply-to: Aaron Ballman (aaron@aaronballman.com)
Document No: N2869
SG Meeting Date: 2021-11-05

Fri Nov 05, 2021 at 1:09pm EST (we had a false start due to being zoom bombed)

Attendees

Aaron Ballman	WG21/WG14	chair
Thomas Köppe	WG21	
Will Wray	(14)/(21)	scribe
Jens Gustedt	WG14/(21)	
Jens Maurer	WG21	
Ben Craig	WG21	scribe
Tom Honermann	WG21	
Philipp K. Krause	WG14	
Hubert Tong	WG21	

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

Agenda

Discussing the following papers:

WG21 P2310R0 (https://wg21.link/p2310r0) Revise spelling of keywords
WG21 P2338R1 (https://wg21.link/p2338r1) Freestanding Library: Character primitives and the C library

P2310R0 Revise spelling of keywords

Champion Jens Gustedt, Scribed by Ben Craig

Hubert: alignas may need more exploration, because the positioning of the keyword between C and C++ is different.

Aaron: Same issue as with noreturn? (yes)

Hubert: Maybe make it an attribute like in C++. Want to be able to write the code in C, compile in C++ for good diagnostics, but still have C code at the end.

Gustedt: Still an alignas keywork in C++?

Hubert: _Alignas in C++ will let you put something in a spot that you can put alignas, in clang. You have to put it at the beginning in C++. You can put it in random positions in C, like between int and unsigned.

Maurer: definitely an attribute lookalike in C++. I think putting the thing in front works in both languages.

Gustedt: In C you have the possibility of including the header, and getting the non underscore thing, and C code could already have that. So this is somewhat an existing problem.

Hubert: thread_local, C++ committee. Difference may not be visible to the user? Whole constinit paper, where this was a not small part of the motivation of the feature. Definitely something that is beyond what should be hidden to the user. Extra cost for implementations where it can be made compatible is unfortunate to make it an extra runtime cost. I will note that it is implementer feedback that it is not possible to make it invisible to the user. You will get linker messages.

Gustedt: How is this different from static.

Hubert: popularity of impl mechanism is reversed. For static, most impls do the initialization up front and program start. For thread_local, all that I know of do deferred init. There's dynamic libraries that confuse things too, but no one really knows how to deal with dynamic libs in the whole program translation model.

Hubert: If you're just a pure C++ impl that doesn't care about C or posix or pthread, then you could implement things up front, but otherwise you need cooperation.

Gustedt: For the user, it looks the same. Only seems different whether there's a ctor or not. Would be good to get that more in sync.

Hubert: You still incur an overhead if you fail to mark with constinit. Right now constinit is not enough, there are cases where C++ still needs an overhead, even with constinit, because there's not enough info about the type. Users believing the C and C++ keywords are the same are misinformed, and if they are different, we want them to look different.

Aaron: Are we convinced that the differences are intentional?

Hubert: The dynamic initialization aspect of thread_local is not an sg1 matter.

Maurer: The mechanics we are discussing are sequential.

Aaron: Do the people who care about this topic is an intentional difference between the languages?

Hubert: This is not a divergence between the two, it's a divergence because C++ has ctors and dtors.

Aaron: That's what I was missing. Would be unfortunate if the two thread_locals are materially differnt.

Maurer: That's already happened, with the header based keywords.

Ben Craig: Will #ifdef provide a consistent result across conforming implementations?

Gustedt: wg14 didn't want to provide a consistent result here.

Ben Craig: The MSVC standard library does an ifdef on most keywords, and errors if any have been defined. The library doesn't want to defend against users doing heavy macro things.

Gustedt: The optional [Change 4] would help mitigate that. If we put that in the constrains section, then redefining those things would be a syntax error. If it's in the semantic section, it would only be UB.

Aaron: Visual studio also defines static_assert in C mode without any diagnostics. Their C code already seems to assert this spelling of static_assert. So some prior art there.

Maurer: One of the purposes of sharing code between C and C++ is header files that compile in both so that you can interop between the two. I see how that helps with alignas. How would that work with thread_local though? I can #ifdef around constinit, that takes me half way there. That doesn't take me all the way there because that goes on the definition and not the declaration. Accesses to the thread_local in remote locations need to assume it is delayed init.

Hubert: constinit can be on declarations. It is most of a solution. It isn't capable of dealing with destruction. If you have a class type that is incomplete or an array of incomplete, and you access it, then we don't know whether or not we need to trigger the registration of its destruction.

Maurer: That can be worked around by users by avoiding incomplete types.

Hubert: Yes, as a guideline. There are ways for users who are aware of the problem to address the issue. Trying to make it more obvious that people might want to notice that there is a difference.

Gustedt: These are problems that already exist because of the headers with macros, so this code is probably out there. In that sense it is orthogonal of the change of these to keywords.

Maurer: I think I agree. We already had the spellings as macros in C. The code looks the same with this paper, except that maybe you can get rid of the header. These seem like real problems and it's good we noticed them. Possibly record these difference in the C compat annex in the C++ standard. But for this paper, I don't think this problem is any better or worse with this change.

Hubert: alignas is very fresh in this meeting, there's a chance we haven't had enough time to think about it. With thread_local, yes, it's an existing problem, and there's probably not too much we can do about it. Maybe there's a method were C deprecates the macro then introduces the new name as a macro.

Gustedt: That would take 12 years

Hubert: Yes. The path would be even longer for thread_local, maybe 15 years. I don't we have the full thread_local picture. The C version has semantic value that C++ doesn't have yet. Or maybe C++ could introduce the C keyword with the positioning relaxation.

Gustedt: Unrealistic to deprecate at this point. For alignas, not an accident that this is a keyword. Would be much resistance to make it an attribute. We could restrict the positioning.

Aaron: In C++, alignas is a keyword, but it's specified as an attribute specifier to put it in the right place and to get the right type system behavior.

P2338R1 Freestanding Library: Character primitives and the C library

P2338R0 was previously seen at the June 4th SG22 meeting this summer, and at an earlier WG21 telecon.

Scribed by Will Wray

Author Ben Craig (BC) shares a slide presentation starting with his 'Freestanding Vision for C++':

"To provide the (nearly) maximal subset of the C and C++ hosted library that does not require OS interaction or space overhead".

This is targeted at kernel, microcontroller and GPU developers.

The aim is to improve the development experience on such targets.

The question for SG22 today is:

- "Do we want to require freestanding C to have the non-locale based wchar.h string functions" (wcslen, etc.) and, if we don't, then:
- "Do we want to add a feature-test macro to make it easier on C++ freestanding implementations"

Other things added (or removed) in P2338R1:

<cstdlib>, <cerrno>, <cinttypes> (removed in R1)

stdlib.h: size_t, div_t, ldiv_t, lldiv_t ...

NULL

bsearch, qsort

abs (only integer arguments, not floating point abs / fabs)

errno.h: #defines, but not errno itself

<cinttypes> is no longer added in P2338R1

(i.e. imaxabs, imaxdiv + related types & functions removed)

<cstring>

string.h: all string functions except for strtok (uses global state)

(Recent C drafts may have already added these).

<cwchar>

wchar.h: wmemset, wcslen wmemcpy, wmemmove wcscpy, wcsncpy, wcscat, wcsncat, wmemcmp, wcscmp, wcsncmp, wmemchr, wcschr, wcsrchr, wcscspn, wcspbrk, wcsspn, wcsstr, wcstok

(note, gets wcstok, as it has no global state, unlike strtok)

(note, there's no w* equivalent of POSIX / C23 memccpy).

In the embedded space, and in other spaces where freestanding is useful, wchar_t is rarely used.

Arguments for adding whcar:

- Although the benefit is lower than for the string.h functions, it is enough to make support worthwhile for use cases that require it.
- It is portably implementable so fits the 'maximal subset'
- There are ecosystems where wchar is present and useful, in particular, Microsoft and EFI / UEFI ecosystems use wchar_t a lot.
- On the C++ side, string_view has wchar_t specialization so it'd be useful to include it in C++ freestanding implementation.
- There's no runtime size or execution time penalty when not used

(some linkers don't do function level or object level linkage, but there are workarounds for these problematic linkers - you don't have to put all functions in same library archive, they can be split into chunks with linker flags as for -lm -pthread).

JM: remark on POSIX env not having everything in a single C library may be a historical accident - e.g. math implementers specialized authors. For embedded setups with broken linkers, unable to do per-object files, they will already have this difficulty to limit size with other C libs.

BC: Wish Rajan was here as this was one of his feedback comments. Agree with JM but Rajan may be able to rebut this remark.

PK: Yes, shame Rajan is not here; IIRC his use case was for a mainframe boot-loader that cannot leave any part of the library out.

BC: Right; think that's what he was referring to. The mitigations might work - will move this to the mailing list for comments / rebuttal.

JM: wonder if a mainframe boot loader is something far away from the user of the C library,

HT: Responding to JM observation that C library itself is already costly, so if it is already implemented in firmware then including whcar could increase cost or baggage.

JM: That's an argument against any expansion of freestanding though.

HT: Unless there are mitigations on the implementation or its toolchain, such as the 'chunked libraries' or different profiles that user must configure if support is needed.

BC: Right, the standard says there should be a mode that is conforming so an implementation can have a mode for full conformance or not.

JM: People have worked around this in the past with more levels of conformance between freestanding and bare metal.

BC continues, presenting the feature-test macro alternative:

__STDC_HAS_WCHAR_H_FREESTANDING_LIBRARY

The alternative (to wchar in freestanding) on C side (especially with C having been more amenable to optional features than C++ has been) is to have a feature-test macro for optional availability of the wchar support (don't have wording for this alternative yet.)

Notable omissions:

- errno
- Many string functions that use errno
- atoi - uses locales and TLS (thread-local storage)
- strtok rand - use TLS
- assert - stderror stream
- - global / TLS data
- Not requiring that C freestanding include exit functions and atomics (as included in C++ freestanding)

PK: Regarding strtok; while you might not like that it uses global state it is part of C freestanding and it would make sense for C freestanding to be a subset of C++ freestanding.

BC: Grudgingly accepts, makes a note to maybe accept this on a next look.

Experience with C libs:

- musl, newlib, uclibc-ng include all C parts, commonly used in embedded
- SDCC includes most, apart from div and whcar
- IBM freestanding was missing bsearch, wchar and errno #defines (with reservations about adding the wchar support)
- Windows kernel C library implements most C additions
- Linux kernel doesn't implement much of it.

Potential C breakages:

- In theory a user can provide their own version of, say, memcpy and as long as they don't include string.h then that's required to work. This could cause problems in terms of linkage, e.g. updating from C17 to C23 but implementers can mitigate with macros.
- C users can't provide <cstring> (UB)
- User could avoid <cstring> & define on memcpy
- C impls can mitigate with macro trickery

JM: Why are we picking on wchar in particular? IIUC this is one part of the proposal, which also includes bsearch, qsort etc.

BC: Because it is an order of magnitude less useful than, say, bsearch.

JM: Two concerns, (1) it is more work, (2) it increases binary size. The wchar versions seem not much more complicated than narrow versions?

BC: If implemented in assembler, as is common, then it could be tricky.

TH: (confirms that wchar is used in MS & EFI ecosystems)

TH: Does anyone know about DSP environments (this has come up in SG16) for DSP, wide string is often just ascii with bytes 16-bit wide. Do you know of systems that do any more complex wide string processing?

BC: I don't know, each seems unique, don't know the common wchar sizes.

WW: As an optional feature, in which areas is C more amenable?

AB/BC: VLA, threads, complex, atomics, annex K, dfp, bfp...

PK: SDCC switched to 32-bit some time ago. Effort was not so much, yes. But how useful is it really? Microsoft already has non-conforming wchar. So do we want to encourage its use?

BC: Agree that communities are generally moving away from wchar.

TH: Comments that WG21 has a paper to make microsoft wchar conforming.

HT: ...Conforming for core language but causing UB in some lib cases...

POLL: Remove wchar.h additions from P2338R1?

Committee	SF	F	N	A	SA	Notes
WG14	1	0	1	1	0	No consensus
WG21	0	0	4	4	0	Consensus against

AB: no consensus for change

BC: On the C side, it's unclear whether to do the macro feature test. That can be decided by WG14.

AB to PK: Would your vote be different if it was an optional feature?

PK: I think wchar doesn't have enough meaning - if it can be anything then what's the meaning. Was meant to represent any code point. Not a good time to add to the standard.

AB: We have a very small wg14 quorum. Others may have a different view. It could make sense to take to WG14.

BC: Will start by raising it on the mailing list. Getting to meetings could be an issue (if not ISO registered).

AB: The reflectors are a good place to start.The WG14 convener is happy to invite subject matter experts to attend.

PK: The paper changes the editorial style of headers; it could add a few pages to what is currently a few lines.

(more discussion of string.h and wchar.h editorial style)

BC: Not opposed to changing. Tried to keep freestanding in one place rather than keep changes local to headers.

BC: on C++ side it's a bigger thing. On C side don't want to change the style, will follow existing style to minimize diffs

(some discussion of floating point functions, BC avoids adding any FP)

End at 2:38pm EST