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

Reply-to: Aaron Ballman (aaron@aaronballman.com)
Document No: N2689
SG Meeting Date: 2021-03-05

Fri Mar 5, 2021 at 1:00pm EST

Attendees

Aaron Ballman	WG14 WG21	chair
David Olsen	WG21	
Tom Honermann	WG21	
Jens Maurer	WG21	
Clive Pygott	WG14 (21)	
Michael Wong	(14) WG21	
Hubert Tong	(14) WG21	
Hans Boehm	(14) WG21	
Mitsuhiro Kubota	WG21	
Jens Gustedt	WG14 (21)	
Miguel Ojeda	WG14 WG21	
Robert Seacord	WG14	
Will Wray	(14) (21)	
Ville Voutilainen	WG21	
JF Bastien	WG21	
Gabriel Dos Reis	WG21	
Ben Craig	WG21	scribe
Steve Downey	WG21	
Nevin Liber	WG21	
Tomasz Kamiński	WG21	
Corentin Jabot	WG21	
Peter Sommerlad	WG21	

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

Agenda

Discussing the following papers:

P2264R0 "Make assert() macro user friendly for C and C++" (http://wg21.link/p2264r0)
P1315R6 "secure_clear" (https://wg21.link/p1315r6)
P1152R4 "deprecating volatile" (https://wg21.link/p1152r4) (tabled due to lack of time)

P2264: Make assert() macro user friendly for C and C++

Champion: Peter Sommerlad

Chair: Aaron Ballman

Ben: Do you have implementation experience?

Sommerlad: No

Ville: Does this paper cover the earlier feedback given on the D revision? You can refactor a static assert into a regular assert that never fires. David Stone suggested an alternative implementation that wouldn't suffer from that problem. There were other ruminations on how to avoid top level commas entirely. Was this paper intended to cover that feedback and address it?

Sommerlad: I covered the feedback that I got from the mailing list in section 4.

Ville: It seems avoidable to have the top level comments, and somewhat perilous to change a static assert into an assert and have that produce an answer that will never fire. That seems like a bear trap to me.

Sommerlad: So changing static assert to assert today is a compiler error, and with my change it will result in something that will never fire? When is that a use case?

Ville: If I change some compile time code to run time code, then I may do that. I would like the paper to address David's feedback there. If you have an assert with condition,string... then that's not beginner friendly, but it ends up asserting the literal. I don't find that beginner friendly.

Gustedt: Can you point to the example of David's implementation? Is it C capable or C++ only?

From Corentin Jabot to Everyone: 12:18 PM

https://godbolt.org/z/87hnss

Gustedt: Although this problem is more severe in C++, it can appear in C with compound literals. I am in favor of the current approach, though the C wording likely needs some work.

Ballman: The implementation Corentin pasted does look C++ specific

Gustedt: Replacing static_assert with assert seems like less of an issue in C

Honermann: One thing that concerns me with VAARGS, I can see a user expecting that each comma seperated thing being evaluated, as opposed to evaluated as is.

Ballman: the macro definition does show up in IDEs

Sommerlad: Seems like QOI to prevent additional top level commas

Ben: assert(expr && string literal) is a common pattern, so we would need to be careful in

Maurer: Probably not correct to say that preventing `, string literal` can be QOI. C spec says that it should accept that.

Sommerlad: It may be that scalar expression doesn't allow a common.

Gustedt: scalar expression has no definition in the C standard

Ballman: An expression that resolves to a scalar type.

Gustedt: This looks like a defect in the C standard that has been there for 30 years

Maurer: Italics scalar could get you somewhere

Sommerlad: Intent is convertible to bool

Corentin: This fixes a lot more issues than it corrects. Not convinced that replacing static_assert with assert is a common issue. Fine for an implementation to diagnose when there is a comma operator in there. I don't see a purposeful use case for having a comma expression in there.

Gustedt: You could say something about what lexical expression is permitted. Scalar in front suggests that this is a scalar value. The comma expression is the last of the list of expressions in C. You could say that this is an assignment expression.

Maurer: That's my suggestion too.

Gustedt: You'd need a paper to fix that too. You'd need to add it to this paper.

Sommerlad: Will need help with the C compat wording style

Ballman: For wording problems, I'd suggest emailing the liasons list.

Corentin: Would the C committee be comfortable with that being compiler magic?

Gustedt: For me, it would be ok.

Ballman: Tough to know how they would feel about it since there are so many implementers not in this call. With chair hat off, WG14 would like to see implementation experience.

Sommerlad: So change the gcc implementation and build a bunch of things and run the tests?

Ballman: Best result is to have someone with deployment experience with delighted users. Just trying things out on your own can tell you if there are any known breakages.

Sommerlad: I'm not sure how existing code would no longer compile with the change. I can see how old code that didn't compile will start to compile.

Ojeda: Extensions for C compilers may interact with whatever is happening in the macros. It's also with whatever magic they are doing.

Sommerlad: Would like a poll to see if we want to do something about this.

From Robert Seacord to Everyone: 12:37 PM

so in C this would be a "something along the lines of the solution presented in this paper"

From Tom Honermann to Everyone: 12:37 PM

Tangent: It might be useful if we had some documentation somewhere regarding how to validate compiler/library changes. For example:

- How to rebuild Debian with an altered compiler/library

- How to rebuild all vcpkg packages.

Ville: Would like to fix some of the kinds of things that show up with std::pair, but this seems to be trading off one set of problems for another, and that doesn't encourage me.

Corentin: I'd like to get compiler magic, but even if we can't, I'd still be in favor of this.

POLL: Should the assert macro be cahnged to be a variadic macro along the lines of P2264R0?

Committee	SF	F	N	A	SA	Notes
WG21	2	7	2	2	0	Consensus, author position was SF
WG14	3	1	0	0	0	Consensus

Corentin: Any statements from the against?

Against: 1) Trying to remove surprises from the language. There are still surprises with this suggestion. These issues are a little worse here in that it takes a weird compiler error and turns it into a subtle runtime error. 2) contracts are coming and would be a better solution in general.

Against: If the approach were a bit more focused on the assignment expression rather than a variadic macro, it may be better. Assert has been around for a while, so I don't see this as well motivated, especially with contracts coming.

Hubert: I thought the poll was on a general direction, which makes the objection based off an assignment expression weird.

From Robert Seacord to Everyone: 12:53 PM

We do not currently have any proposals to add contracts to C23

From Corentin Jabot to Everyone: 12:54 PM

If we add them to c++, c++ won't need assert, so c doesn't have to care about c++ types with comma

From Miguel Ojeda to Everyone: 12:54 PM

Isn't "along the lines of" is about the idea rather than the implementation?

Ville: It's nice to have a poll that says "along these lines" with a hope for a future change, but I can't support the paper until I see that change.

WG14 N2631, P1315R6 secure_clear

Champion: Miguel Ojeda

From Ben Craig to Everyone: 01:02 PM

Any consideration into modifying C 5.1.2.3 Program execution p6? There is similar language in C++'s http://eel.is/c++draft/intro.abstract#6

Seacord: Could this function indicate errors? It currently says returns a pointer to s. For null pointers, that works out. Maybe size of 0 is considered an error? Maybe rsize_t type that was developed for Annex K. Really large values would be considered errors too. Anything bigger than unsigned char max could be an error too.

Miguel: We want the same as memset. If we want to change the interface, then that's fine. Not sure if we want to go into that. memset doesn't currently return an error.

Seacord: You have the possibility through returning null.

Miguel: If we do something different than memset, then that could give the wrong impression.

Seacord: I think everyone that uses memset where you might take the return value as an argument to another function call.

Miguel: Maybe have termination semantics if you do something like have a too high value of 'c'.

Seacord: Can't do that, because of embedded systems. You could add error indicators here without changing existing code.

Miguel: What about errno

Robert: I don't think we add errno to anything on purpose anymore. I think that's just a legacy mechanism that we still support.

Ballman: In the paper, one of the things that it mentions is keeping the performance similar to memset. Adding error checking would be in tension with that.

From JF Bastien to Everyone: 01:10 PM

No errno, or error checking here please.

Hubert: I think the paper author said that that is sort of going away.

Gustedt: I think I would be against changing anything with the memset semantics. Would really mix people up. I like Alternative 2. I think Alternative 3 is too detailed and not quite correct. Adding a volatile somewhere could enforce a side effect on the byte. I think you should also talk about bytes and not characters.

From JF Bastien to Everyone: 01:10 PM

No volatile please, because that prevents multiple writes to the same byte.

Boehm: I don't think barriers make sense in this context, I see code like that in the kernel and it doesn't make sense to me there either.

Miguel: Users in the wild will put barriers in their projects to get this effect.

Boehm: I don't think that's guaranteed to work, though it might work on some platforms.

Boehm: Does this do anything about passwords getting paged out to disk? This won't help in the case where the password is paged out, and it can sit on disk for hours. I think the specification should say whether that is intended or not. If this hits paging, then it will have a substantial performance impact. Specifying that the performance is similar is good, and helps suggest that dealing with paging isn't an allowable implementation strategy.

Miguel: We want to leave paging as QOI. Can't talk about the memory heirarchy in the standard. We could put it as a recommended practice. We are trying to mimic what people do in the wild.

Seacord: Instead of saying "the purpose of this function is to" I would say "A possible use of this function is...". I wouldn't want to tell people this is the only reason to use it. Also, it feels nonnormative so should be in a footnote as well.

From Ben Craig to Everyone: 01:17 PM

I believe that in C++, footnotes are normative

(no idea about C)

From Ville Voutilainen to Everyone: 01:17 PM

no

in C++, footnotes don't do anything

From Ben Craig to Everyone: 01:17 PM

ok, guess I'm wrong then :)

From Ville Voutilainen to Everyone: 01:18 PM

notes, examples, footnotes, they're comments, an implementation can turn them into whitespace

From Robert Seacord to Everyone: 01:18 PM

Also agree with Jens that it should say "first n bytes" instead of "first n characters".

Boehm: In alternative 2, replacing spilled registers with paging will address my concern.

Ben: What about the abstract machine wording?

Hubert: That all falls back to implementation definedness. The wording in alternative 3 is stronger than the abstract machine clause

Ben: I think that may be too subtle, but if it works, then ok.

Hubert: The C standard says that you really shouldn't remove needed side effects, but it doesn't say why they are needed. The alternative 2 definitions gives a false sense that things are more defined than they are. If we like the QOIness of alternative 1, then fine. At least it is up front about it. Alternative 2 doesn't give a good impression about that. Alternative 1 recommended practice part I hear is going away, and I don't like saying too much about performance, because I don't think it matches that. A user calling memset may have the compiler not call memset and inline code, which is different than calling out to a memset function.

Miguel: I think Peter Seawell(?) had similar comments. I think he was saying that he preferred the first alternative, just to avoid needing to discuss the abstract machine ramifications.

Maurer: I'm with Hubert here. It's hard to specify this function. There's always the question of "is my implementation conforming", and ideally the standard would be razor sharp in defining that. So far, none of the alternatives satisfy that. Alt 2 and 3 don't stop dead store elimination in hostile implementations. A hostile implementation could clear the bytes, then put the data back. Alt 1 speaks to that a bit more. That implies that the compiler or implementation must look at register spills and paging, and we probably don't mean that. I think it's close to impossible to properly integrate that into the abstract machine. Existing memset_s probably has the same problem.

Gustedt: I like Alt 2 because it says something about sequencing. I think it's important to say what the side effects of this function does. Saying that everyone who sees that call observes the side effects is important.

Maurer: "needed side effect" doesn't make sense.

Boehm: Flushing cache and backing disk things need to be clarified. As a user, I need to know how to use this. If it leaves that unclear then I don't know.

From Hubert Tong to Everyone: 01:29 PM

+1 doesn't make sense to me too

Ville: Not sure if any of these attempts have meaning in the memory model.

Miguel: SG1 looked at the paper, but they didn't see these alternatives.

From Nevin Liber to Everyone: 01:30 PM

I don’t see how we can ever specify that we clear a paging backing store, as that is in the domain of the OS and not the programming language.

From Robert Seacord to Everyone: 01:30 PM

This is for C

From JF Bastien to Everyone: 01:30 PM

I like how we rehash the same discussions every time we discuss this paper.

From Jens Maurer to Everyone: 01:30 PM

We can't, and that's the point: We can't specify this properly, in the sense that Hans described.

From JF Bastien to Everyone: 01:31 PM

SG1: We don't see a specific SG1 concern for what we presume is a single-

threaded API. If this paper intends to make secure_clear resilient

to UB in C++ then data-race UB is but one of the kinds.

SF F N A SA

No objection to unanimous consent

From Jens Maurer to Everyone: 01:31 PM

JF Bastien: To make stink, you need to object to a paper at every level. That's what I'm doing.

From JF Bastien to Everyone: 01:31 PM

See all prior discussions here: https://github.com/cplusplus/papers/issues/67

Ville: The problem is that if it isn't an abstract machine effect, then the desired affect won't be clear to a C++ optimizer.

Miguel: If the implementer knows the intent, then it doesn't matter what the abstract machine says. This can be implemented today without optimizer changes.

Ville: the compiler just needs to mark it as not subject to IPA. Do we have feedback from implementation vendors?

Miguel: We have some votes from the C committee

Ville: I'd like to know if we have that feedback on the C++ side.

From Nevin Liber to Everyone: 01:35 PM

If we don't want to rehash discussions, the paper should answer those questions. It shouldn't require archeology.

JF: I've maintained a few implementations of memset_s and haven't had difficulty getting it done the right way, even though it doesn't follow the rules of the abstract machine. If you know your compiler, then you can implement it without changing the compiler. You can do brittle things like having external TUs or assembly, but an implementation owns its own brittleness.

Hubert: The reason there are alternatives is that the first alternative is all intent, and people weren't necessarily happy with all intent. Some of the more wording minded people realized that alt 2 pretends to do more, but doesn't. Alt 3 came up, as it at least forces the implementation to say something. If we decide that Alt 1 is good enough past Ville's concerns, then I think the spec isn't only talking to implementers though. Also used by users to understand what the language is and does. I think both alternatives 1 and 2 leaves those people high and dry.

Ville: Specify the intent, don't bother with guarantees, then say that the side effects are implementation defined. That's probably the best we can do. The intent will make the implementers stop and ponder, and force them to apply the known techniques to match the intent. It's not the best of all worlds, but it's avoiding the worst.

From Jens Maurer to Everyone: 01:42 PM

Ville: sounds good

From Hubert Tong to Everyone: 01:42 PM

+1

Miguel: Alternatives that aren't intent only will have some experts from somewhere that don't agree with the wording. Main issue with alternatives 1 as a user is that some users will need more guarantees.

From Corentin Jabot to Everyone: 01:43 PM

"The intent is clear" is a great pun 👍

Ville: would suggest amending the semantics have implementation defined semantics, and to train the users to look at their manuals.

From Robert Seacord to Everyone: 01:45 PM

but then it needs to be documented

From Jens Maurer to Everyone: 01:45 PM

Yes, that's the point.

From Robert Seacord to Everyone: 01:45 PM

Inside C joke. We have a vendor who doesn't want to document anything.

From Jens Maurer to Everyone: 01:45 PM

The detailed semantics are per-implementation, and the user discusses with his implementation whether he likes the function or not.

Maurer: As long as we make it clear that this isn't a portable function, then great.

JF: There are some differences in the C++ and C APIs regarding objects vs. values.

From Robert Seacord to Everyone: 01:52 PM

the semantics that are defined should remain defined

the semantics that aren't defined are implementation defined?

From Ville Voutilainen to Everyone: 01:53 PM

I'd suggest we try to split that hair separately

From Robert Seacord to Everyone: 01:53 PM

ok

From Jens Maurer to Everyone: 01:53 PM

The point is: there is just a statement of intent, but not really any effect in the sense of the abstract machine.

JF: The C++ committee didn't want to provide a specific value.

Miguel: The poll was for the plain function. There were some people that wanted the function without the value. C committee wants the parameter. Might end up changing the C++ one.

JF: You will get significant push back unless you say why it is important to be able to provide the value.

JF: We haven't discussed the point that these functions would have different interfaces.

Maurer: Totally different question. T& question wasn't on the plate today.

JF: The current polling suggests that I approve of alternative 1 if I want alt 2 and alt 3 were dropped.

POLL: Should P1315 memset_explicit use alternative 1 plus a statement that the semantics be implementation-defined as a statement of intent rather than an effect on the abstract machine?

Committee	SF	F	N	A	SA	Notes
WG21	5	5	0	0	1	Consensus, author position was SF
WG14	2	2	0	0	0	Consensus

SA: To clarify my vote: I’m strongly in favor of a single alternative, which states that the semantics be implementation-defined as an intent rater than an effect on the abstract machine. I am not in favor of the alternative 1 API, because I don’t believe that it should contain the `int c` parameter.

Wrapup

Hubert: First Friday of April is the start of Holy Week, we should reschedule.

Aaron: I’ll reschedule and send an announcement, thanks!

End at 3:02 pm EST