Clang C/C++ Language Working Group Meetings

Meeting Details

Community Calendar
https://calendar.google.com/calendar/u/0/embed?src=calendar@llvm.org

Google Meet
Clang C/C++ Language Working Group Meeting
First and Third Wednesday of each Month · 11:00am – 12:00pm Eastern Time (UTC-5 when DST is off)
Google Meet joining info
Video call link: https://meet.google.com/ifi-uqto-kjw
References
Clang - C++ Programming Language Status
https://clang.llvm.org/cxx_dr_status.html
https://clang.llvm.org/c_status.html
https://clang.llvm.org/c_dr_status.html
https://github.com/orgs/llvm/projects/30 – regression tracking

Meeting Agenda - July 1st, 2026

- Clang 23 is scheduled to branch on July 14th
    - Optional 22.1.9 release did not happen
    - Corentin: expansion statements?
    - Aaron: I think nothing is on fire, so it can wait until 24. Or we think reviews are far enough to merge?
    - Corentin: I was about to start approving first PRs
    - Ambrose: my stance is that I have 2k lines of tests. I might have missed something, but I think my implementation is more complete than GCC
    - Aaron: are the first patches enable the feature?
    - Ambrose: no, they have to merged all together
    - Corentin: we don't need to be concerned about refactorings — they just move things around
    - Ambrose: there are things I need to address from your review
    - Ambrose: there is that RAII object I use to parse the statement to track some context
    - Corentin: I don't mind that, I've been thinking about replacing source locations with a pointer
    - Ambrose: I did that
    - Corentin: please ping me, so that we're not wasting time
    - Ambrose: let me check whether I pushed that
    - Ambrose: we also discussed whether we need to track source location of opening and closing parens
    - Corentin: you convinced me we should
    - Corentin: we have the location of the template keyword
    - Aaron: normally we only track things we need, but I don't oppose to tracking this upfront. Do we have a problem of lots of instantiations?
    - Corentin: we kind of have a lot of instantiations
    - Aaron: either is defensible
    - Aaron: if we think it's ready for 23, I don't object. But I wonder how much time Ambrose have after the branch date
    - Ambrose: I do have time to address concerns
    - Corentin: it would be useful to land c++2d PR
    - Ambrose: I was waiting for your approval
    - Aaron: I approved it
    - Corentin: let me approve it right now
    - Corentin: Aaron, I'm going to add MSVC-specified language mode tomorrow
    - Ambrose: after the patches land, I'll put a patch for iterating expansion statements with token injection. Not going to be a lot of code
    - Aaron: makes sense
    - Aaron: Muhammad is asking about if declarations
    - Aaron: I've talking with Erich about this. We didn't have time to dig into this, but we were surprised by attribute handling. It seems to paper over preexisting bugs.

    - Corentin: I can try looking at it
    - Muhammad: C++11 and C23 attributes have something that is different from GNU attributes. Handling code expects C23 attributes, not general attributes
    - Aaron: we saw your analysis. We believe you, we just don't understand why the status quo the way it is. GNU attributes can slides around, and we want to make sure the parser doesn't end up in invalid states.
    - Vlad: rushing all the analysis in the two weeks we have left doesn't sound a good idea
    - Aaron: yes, we don't need time pressure here, but I understand that Muhammad has his own timelines
    - Corentin: I'll try to take a look.
    - Vlad: how much time is there for parser issues to surface?
    - Aaron: we'll see them in 48 hours
    - Vlad: how much time Muhammad has after the branch date?
    - Muhammad: I have a midterm evaluation at the beginning of August. I don't have strong opinions whether we should land this in 23
    - Corentin: is anything on fire?
    - Aaron: I haven't seen anything
    - Shafik: me neither
    - Aaron: I was wondering if we have a higher rate of regressions on 23
    - Shafik: that would require more work. I haven't seen anything outstanding while triaging. Matheus' patches caused disruptions, but that was expected.
    - Aaron: good to know. If we see a bunch of issues that come back to those patches, reverting them will be painful
    - Shafik: yeah, that wouldn't be my suggestion either
    - Shafik: fuzzing folks are catching stuff. There are some organic issues which might be more concerning
    - Aaron: let's cross our fingers and hope for the best. Otherwise we'll have to be more careful about those big refactorings. That stuff is complicated
- P3899R3 "Clarify the behavior of floating-point overflow" was adopted in Brno as a defect report
    - The paper aligns the wording with the GCC 15 behavior (there is a table in the paper)
    - Aaron Ballman raised last-minute concerns over making it ill-formed for constant expressions to produce non-finite results from finite values, because this is what windows system headers and users to with INFINITY macro
    - New information: we implemented the required behavior in Clang 8 but reverted it in Clang 9 (review)
    - Aaron: infinity is in the range of representable values in IEEE, and we intentionally support it.
    - Corentin: I agree with the idea of a degradable error, but I'm not sure I like the direction
    - Aaron: there might be disagreements with C23

- Shafik: this is based on FP_EXCEPTIONS and other implementations, so that's why Core felt it was reasonable
- Aaron: there were concerns that this will break code, but then there were concerns that this will be painful to support
- Ariel: is this based on IEEE or is it going to be hardwired?
- Aaron: hardwired
- Ariel: how is this going to work for formats where infinity is not representable
- Aaron: …
- Ariel: then it's dependent on the format
- Aaron: then you can't generate infinities during compile time in C++ now
- Ariel: you said earlier that you only can do that in initialization
- Aaron: that's what WG21 wants. We implemented that in Clang 8, then reverted. We'll break code on platforms with infinities. I'm surprised that WG21 wants something else from IEEE
- Shafik: Hubert and Davis were a part of this conversation. Jan mentioned that he was copying a lot of things from C
- Shafik: we can write to the reflector
- Vlad: I have already done that
- Aaron: WG21 doesn't seem to willing to back down, and I don't want to break user code
- Shafik: so we're going to have diverging constexpr behavior.
- Aaron: yeah, to me the pedantic diagnostic makes sense here
- Aaron: not many people compile with pedantic
- Corentin: the way we decide whether something is a constant expression is by examining the presence of diagnostics
- Aaron: then we should ignore the paper. WG21 did not give us any rationale to change
- Aaron: we should push back
- Corentin: do you want everyone to be compatible with Clang?
- Aaron: that would be fine by me. Or implementation-defined. If your constant evaluator can't handle it…
- Corentin: you want implementation divergence?
- Aaron: if we were to force a change, we should force Clang behavior, because it's useful
- Aaron: in math you can get infinities
- Corentin: but the infinity here is basically an overflow. THis is inconsistent with runtime
- Aaron: I'm not convinced we'll ever get to the point when constexpr and runtime floating-point results will be the same
- Aaron: I want users to have a way to get to the old behavior. If we want a flag, so be it, even if it would be a silly use
- Shafik: the problem was that this is not a portable code, and this is not useful for our users anyway
- Aaron: we don't have good metrics on how many people will be broken by this

    - Shafik: the first things to say is that we have concerns, then solicit opinions from GCC and EDG. The current situation is not great
    - Aaron: how do you produce a constexpr cmath without producing infinity?
    - Corentin: this thing does not preclude infinity
    - Aaron: [points out to the wording saying that infinity is not a core constant expression]
    - Vlad: tried to point it out, but Jan was told it's fine
    - Corentin: the conversation I was a part of was about macro in windows headers. If we have more issues, it's problematic
    - Vlad: all the arguments, except for this new argument about constexpr cmath, are already on the thread
    - Corentin: the constexpr cmath argument is a much stronger argument
    - Aaron: we pointed out on the thread that we implemented and reverted it, and we were ignored. I'm tired of this.
    - Shafik: the discussion in Core wasn't contentious, so this point wasn't really represented
    - Shafik: the arguments on the reflector weren't particularly convincing
    - Aaron: we shouldn't be breaking people. WG21 shouldn't need convincing to not break people
    - Shafik: fair. I'm pushing back on CWG intentionally breaking people
    - Aaron: fp is inconsistent
    - Shafik: you'll hit this in C soon
    - Aaron: C should be fine, but I don't know what GCC does
    - Aaron: WG14 pays exclusive attention to FP, whereas WG21 just starts
    - Corentin: we are saying it's well-defined but not at compile time, and I'm not convinced it's justified
    - Shafik: division by zero exists
    - Shafik: with constexpr cmath we'll start seeing inconsistencies
    - Aaron: I think GCC C frontend does the same as Clang: https://godbolt.org/z/dePY1YdKK (example was bad, different example showing more confusion: https://godbolt.org/z/Y74x7faW1)
    - Shafik: this might be a new information
    - [confusion about the behavior of compilers]
    - Aaron: I no longer blame WG21. This is a mess
    - Aaron: no constexpr is fine
    - Ariel: it does division
    - Shafik: it's core language UB
    - Aaron: we'll ignore the paper for the moment. Stakeholders are off in July, August might be a better time to continue the discussion
    - Aaron: we'll continue when the stakeholders are present
- On the forums (DID NOT GET TO THIS):
    - Revisiting commit-access criteria (2026) (thread)
        - Fangrui suggests to move back to "substantial" bar for getting the commit access

        - Several people suggested that these days it would make more sense to focus on review activity instead of PR authorship
    - [FileCheck] Improving input dump readability (RFC)
        - A nice quality of life improvement
        - The associated PR has landed

Meeting Agenda - June 17th, 2026

- Clang 22.1.8 is out
    - Clang 22.1.9, if necessary, is scheduled for June 30th
        - Corentin: I was away for WG21 meeting, and I'm not aware of regressions.
    - Clang 23 release branch is scheduled to be cut off on July 14th
        - What are the priority PRs and work items targeting 23?
            - Ambrose: would be nice to get expansion statements in
            - Corentin: I need to review everything again
            - Ambrose: I have couple of items to fix after your last review, but no major works
            - https://github.com/llvm/llvm-project/pull/169680
        - Blockers / Regressions?
            - Corentin: I haven't looked at whether we have major regressions. We should do that before branching
            - Shafik: nothing off the top of my head, but I'll check
            - Shafik: https://github.com/llvm/llvm-project/issues/202957 a regression about concepts
            - Shafik: https://github.com/llvm/llvm-project/issues/175831
- WG21 met last week in Brno
    - P3367R4 "Constexpr coroutines" was not voted in (yet)
        - The paper didn't get the final nod from LWG
        - Corentin: the wording was fine, so Core approved. But every implementation said they will object during plenary, so fireworks were expected. It's going to come back in brazil. In the meantime I saw Tim saying that we will be able to turn the new interpreter this year. So the timeline is shorter. We might only have to implement constexpr coroutines once in the new interpreter.
        - Shafik: curios what he's basing this timeline on. He'll have to put up an RFC
        - Vlad: it will be turned off at least once
        - Shafik: likely more than once
        - Vlad: yes
        - Corentin: he said one year
        - Shafik: so next summer. Later than I understood. We need to have a formal discussion. I was surprised that my comment in Core was picked up by other implementations. When EDG folks said they will raise objections in plenary, I said I'll support that. Jens asked us to write Guy so that he's not surprised. He asked us to write a paper to sustain our objections
        - Corentin: Timm says we can switch on this year. EDG also has interpreter, so for them it should be as difficult as for us

        - Shafik: during conversations in hallways, an alternative approach might be viable
        - Corentin: if we'll get an interpreter within a year, we'll be in a good position. When MSVC says they need a person-year, it means they will not do it
        - Shafik: GCC did not object. Waff3x said he will support it. So my understanding is that they will implement it. But MSVC is not going to. EDG folks thinks the approach is viable, but they will need to check.
        - Vlad: Hana's suggestion seems to suggest to split coroutines in the frontend, like GCC does. We need to talk to Iain Sandoe
        - Corentin: we should spend more time on quantify the complexity before brazil.
        - Vlad: who is going to do that?
        - Corentin: we need to have a meeting with Iain and Timm
        - **Action item: Corentin will set up a meeting**
    - P2719 "Type-aware allocation functions" is under CWG review
        - A lot of progress was made on the wording
        - Corentin: we should make sure the new wording corresponds to reality
    - P3596R3 "UB and IFNDR annex" was voted in
        - Congratulations to Shafik!
        - Shafik: I hope it will be useful
        - Hubert: it has already been useful
    - P4101R1 "Consteval-only values" was voted in as a DR
        - Replaces the unimplementable notion of consteval-only types with consteval-only values
        - Corentin: this changed the reflections. Thankfully, we're not affected, because we haven't implemented it
        - Shafik: Davis was arguing for leaking meta::info, but lost
        - Corentin: me too, but Barry was too convincing
    - P3125R6 "Constexpr pointer tagging" was voted in
        - `tagged_pointer()` returns `cv void*`
        - Corentin: it requires a fair amount of builtins to make it works, but we can do it
        - Hubert: I'm fine because it proposed builtins
        - Hubert: in constexpr land you're allowed to convert from void* and back
        - Corentin: you can convert from void* to object pointer only if there is something there, right?
        - Hubert: yes
        - Hubert: the naive interpretation is when you have tag with zeroes, then at runtime the representation matches a real pointer. You might expect to be able to cast it. Corentin pointed out that such a pointer is still invalid, and we don't need to make it work.
        - Corentin: I'm not sure what the wording says, though

        - Hubert: in this meeting we again changes constexpr, you can have two different pointers with the same representation
        - Hubert: this is how Davis' paper works
        - Corentin: speaking of which, do we need to do anything w.r.t. his pointer paper
        - Hubert: yes. You are allowed to move from one past to end to the object that is really there. We need to propagate provenance less when copying thing
        - Vlad: can people traverse stack variables now?
        - Hubert: only if their addresses were exposed.
        - Shafik: can you open a bug report about this?
        - Corentin: the changes that we need are mostly LLVM, right?
        - Hubert: yes
        - Corentin: so we're again running into an issue that we need middle-end people interested in C++ conformance
        - Hubert: especially if we need more operations. Regarding memcpy, frontend won't be doing anything.
    - P3899R3 "Clarify the behavior of floating-point overflow"
        - Constant-evaluation-wise, aligns the behavior with GCC 15 and FP exceptions at runtime
        - Aaron Ballman raised last-minute concerns over making it ill-formed for constant expressions to produce non-finite results from finite values, because this is what windows system headers and users to with INFINITY macro
        - Corentin: we can recognize the macro and use builtin
        - Vlad: Aaron shared a link: https://godbolt.org/z/f5bTW4j3W
        - Vlad: we tried to do this in Clang 8, but had to revert in clang 9. I tried to pinpoint the commit, but wasn't able to find the trail for additional information
        - Corentin: I think overflow is worth diagnosing, I think
        - Joshua: the paper is going for GCC 15 column and not FP_EXPCEPTION column, as discussed in SG6
        - Corentin: was the potential breakage discussed in SG6?
        - Joshua: no
    - Corentin: I updates the cxx_status page, a lot of papers
- "Implementing P3666R4 Bit-precise integers, in libc++" (thread)
    - Jan is asking libc++ to provide a macro to opt out of library support for `_BitInt`, because LEWG decided they don't want that (yet), because it's a lot of work and there are outstanding questions
    - Libc++ maintainers don't want to flip-flop as LEWG makes their minds, but consider dialing back if C++29 will end up with no support for `_BitInt` in the library
    - Shafik: I find myself supporting Louis' stance. This is encroaching on implementers' prerogative. Jan's position is not convincing

    - Corentin: I wasn't under impression that this is complicated, but apparently LEWG was. Louis stated on the thread that it's not really hard. Maybe it will be more problematic for other implementations
    - Vlad: they also under impression that is_integral_v is going to break users
    - Corentin: they forgot that thye have policy that allows them to extend type traits
    - Vlad: it wasn't brought up
- "Upstream infrastructure for AI reviews of PRs" (RFC)
    - Started optimistically, but lately has been receiving push back from community members who are not fond of AI
- -finput-charset
    - Hubert: non-draft version is being cleaned up, going to publish on thursday
    - Corentin: going to 24?
    - Hubert: it's going to be surprisingly small, might make it into 23
    - Corentin: I was the one to propose the driver flag
    - Hubert: you can make up your mind about -finput-charset when it comes up
    - Corentin: my understanding is that it's very localized
    - Corentin: are you involved in pragma copyright?
    - Hubert: yes, I am. It's an IBM feature, so no one else should be affected
    - Corentin: does it targets 23?
    - Hubert: it might
    - Corentin: I'll take a look
    - https://github.com/llvm/llvm-project/pull/178184

Meeting Agenda - June 3rd, 2026

- Clang 22.1.7 is out
    - 22.1.8 is scheduled for Jun 16
    - Any blockers people are seeing for Clang 23? Branch date is July 14th
    - Corentin: would there be .9?
    - Vlad: if needed. Look at the community calendar
    - Shafik: Corentin, we have several concepts-related bugs
    - Corentin: I'm working on them, they are not easy bugs. There will be bugfixes in .8
    - Shafik: we need a label for the next version
    - Vlad: is the project complete with respect to searching for regression labels? e.g., regression:17
    - Shafik: possible, but it should be relatively complete. Possibly less complete for older issues.
    - Aaron: I have not seen anything blocking 23 release
    - Corentin: I want more eyes on expansion statements
    - Shafik: can you ping me on that? I'll prioritize it
    - Corentin: I'm also trying to review `-fexec-charset`. I think we can't expose it in the driver in 23
    - Aaron: cc1 flag?
    - Corentin: yes
    - Aaron: but the patch is far along to not cause stability issues?
    - Corentin: yes
    - Hubert: There is an adjacent patch for -finput-charset, but it's a draft and has some changes that will be coming.
    - Corentin: I won't have time to look at that until after the WG21 meeting
    - Hubert: it's good, because we'll clean it up in the meantime
    - Corentin: can you review -fexec-charset PR again?
    - Hubert: I also have WG21 meeting next week
    - Corentin: over the next few weeks is fine. If you can't it's fine
    - Hubert: we had an update for preprocessing is never undefined. Maybe worth discussing?
    - Aaron: my understanding is that the author has to justify what they want to strengthen and why. Author seemed to walk away, at least for now.
- WG21 meetings happening in Brno next week
    - Expect less review bandwidth
    - Availability post: https://discourse.llvm.org/t/maintainer-absenses-for-wg21-brno-meeting/90959
    - Timm has some concerns on P1839R7; we don't have a way to model that currently
    - Aaron: I'll be around as usual. I'll be attending WG21 meetings on demand
    - Aaron: there is P1839R7 scheduled in EWG
    - Shafik: Timm has concerns

- Hubert: is this about consteval?
- Shafik: yes, his concerns are about consteval
- Hubert: we have rules to prevent this in consteval
- Shafik: let me read it again, because I thought they want to make it work in consteval
- Vlad: Timm's concerns tie in to the pointer tagging paper. We do not model the object representation of values in constant evaluations. That's what Timm objects to - modeling that level of detail.
- Corentin: we should look at EWG schedule
- Aaron: Vlad mentioned the paper about reflection of attributes. If they decide to model it as implementation-defined for vendor attributes, we won't be able to add new implicit arguments to existing attributes. We also have attributes you can't spell in source
- Ambrose: you mentioned vendor attributes. what about assume
- Aaron: they are punting on hard problems
- Ambrose: then we say you can't reflect over them
- Vlad: we also have problems with ignorability of attributes
- Aaron: it's not like I question the motivation, but they need to do homework. Constinit is a good example. I have serious implementation concerns.
- Vlad: will you attend EWG for this discussion
- Aaron: yes, poke me
- Corentin: most of use cases copy the attributes, but the paper doesn't propose that
- Aaron: even the equality testing is problematic. Easy cases are simple, hard cases are really hard.
- Hubert: everything wonderful about pointers is coming back, including non-deterministic pointer provenance. But there is no new information for this forum. We'll find IR semantics for new semantics.
- Shafik: have you seen compile-time messages paper?
- Hubert: not the last version. We only discussed it once in Core. The design description of the mechanic was unclear. I'm not worried about it from implementation standpoint, but it needs to get through standardization first
- Shafik: I took a look at coroutines. It's a big implementation lift.
- Corentin: I'll object in plenary
- Ambrose: we should reject this until we transition to bytecode
- Ambrose: me and Aaron talked about exceptions. It's possible to implement it in current evaluator, but it's an expansion-statement volume of changes
- Shafik: do you have something I can bring to Core
- Ambrose: we used to return false when we fail, but now we can throw exceptions. We can allocate exception slots, but the whole thing is a lot of work. It's not a problem when exceptions report error, it's a problem if someone tries to handle them.
- Aaron: old constexpr engine tries to update the value in-place, but this is novel control flow. You need full exception handling

- Ambrose: you need handle break and continue explicitly.
- Hubert: it's possible to observe that GCC does single-pass search for the handler in constexpr land
- Aaron: so they hit similar problems
- James: what happens with exceptions disabled
- Ambrose: there are two options. We discussed it with GCC people, and we decided that -fno-exceptions should disable exceptions in both consteval and runtime
- Corentin: senders and receivers depend on constexpr exceptions
- Ambrose: does anyone tries to catch them?
- Corentin: to report failures with better error messages. And for that you need constexpr exceptions. Also catching, so that error is reported in the right place
- Aaron: in S/R you can restart the evaluation
- Corentin: S/R is runtime thing, it's so complicated in terms of templates. Error messages in exceptions save you from pages of useless template errors.
- Aaron: would it be fair to say that they should've used concepts? Or, more importantly, can a user recover from them?
- Corentin: I don't know
- Aaron: as Ambrose said, we can only implement throw
- Corentin: they catch exceptions and emit error messages
- Corentin: when we're switching to bytecode?
- Aaron: the last time I talked Timm said several years
- Corentin: then we need to implement them before switching
- Aaron: if someone comes with a patch, sure
- Ambrose: I can implement it in the old interpreter if we really need it.
- Aaron: we should talk to Timm. we are already at the point when people are obligated to work with bytecode
- Corentin: for coroutines I agree there is no point in implementing it in old constexpr
- Shafik: should we tell people to not bother with the old constepr engine and only work with bytecode?
- Corentin: the status quo is to do work in both. We can't remove the old one right away.
- Shafik: for PoC bytecode is what matters
- Aaron: but when it's upstreamed, old constexpr has to be covered
- Corentin: bytecode interpreter is more constrained in some ways, so it should be doable
- Shafik: yes, it's harder to hack to do funky things
- Vlad: different topic, I'm reviewing the wording for type aware allocators which is currently in core. The wording will improve.
- Corentin: this is already in Clang, right?
- Vlad: yes, but there may be changes in diagnostics. Oliver will have some trips to EWG, some of what was forwarded aren't really working

    - Joshua: one of the paper is about annex F to C++. annex F is IEEE754 conformance: https://www.open-std.org/jtc1/sc22/wg21/docs/papers/2026/p4212r0.pdf
    - Hubert: this is still in SG6?
    - Joshua: yes
    - Hubert: do you know if it's like C comes with macros?
    - Joshua: it's not that far along
    - Ariel: they don't leak into the language
    - Joshua: i'll try to make sure it doesn't like into the rest of the language
    - Aaron: annex F is restricted to IEEE754, right?
    - Joshua: yes
    - Hubert: are we going to defined a part of C standard library?
    - Joshua: no wording yet.
    - Corentin: I found non-public EWG agenda. We'll be looking at CWG3178.
    - Hubert: yes, this is in EWG to know if they are motivated to do one thing or another.
    - Corentin: I have implementation concerns. We don't want to parse anything there
    - Hubert: sure. We need to check what module grammar is, because it's already tuned for fast scanning. Maybe it's doesn't even need tokenization
    - Corentin: even if non-module scenario it forces you to tokenize everything.
    - Hubert: yes
    - Corentin: but we're not doing this but we're still conforming
    - Aaron: I think Michael Spencer and someone else recently did some work in this area in the last three months
    - Hubert: there is not much you can do there. We might be taking some shortcuts, but I don't think you can avoid tokenization entirely
    - Aaron: I also need to take a closer look, because there is a lot of changes to clang-deps-scanner
- stdatomic.h
    - Corentin took an action item to schedule a Clang Area Team meeting between llvm-libc, libc++, and Clang folks to figure out what to do here
    - Corentin: after the WG21 meeting
- Requiring C++20 Modules Serialization Tests for New Features (RFC)
    - Aaron asked Chuanqi to write documentation about how to write serialization tests
    - Aaron: Chuanqi is in the process of writing the docs: https://github.com/llvm/llvm-project/pull/200994
    - Aaron: this is a request for reviewers: if you see changes to AST nodes, be sure to include him in the review so that he can verify the validity of tests
- GCC2 ABI (RFC)
    - Haiku wants upstream support for GCC 2.95 ABI for the features that compiler supports
    - Scoped to codegen; no semantic analysis changes

    - James suggested to keep this out of the compiler entirely: provide a shim library instead, so that old applications that need GCC2 ABI can interface with newer system binaries that use modern ABI
    - Vlad: they were able to complete 3-stage build
    - Corentin: that's something someone need, I don't hear anyone else need it
    - Aaron: Haiku wants it, and I hear carbon folks are interested in separation for ABI
    - Corentin: I think Carbon folks want an ABI library, GCC2 patch do not do that
    - Vlad: they are going to provide buildbots, IIRC
    - James: I'm not sure even Haiku wants it. It might be early to decide whether we want to accept it
    - Matheus: I think we don't need a buildbot, because it's small and we might use cross-compile
    - Aaron: makes sense. I'm not insisting on CI, but I want it to be tested. Ideally, we shouldn't accept it
    - Aaron: my concerns about new language features were addressed
    - Corentin: i'm very opposed to supporting new language features there
    - Aaron: even the features that do not have ABI impact?
    - Corentin: yes, it's too much work to pick the features to support
    - Matheus: how long are we going to support various quirks we already have in e.g. mangler for old compilers?
    - Hubert: they didn't clarify what they are going to do on the periphery, like debug info.
    - Hubert: I wanted to know the status of constexpr cmath. The author was actively pinging
    - Tue: I also published a PR about constexpr cmath with just APFloat. I noticed that we're not do much checking for exceptions. I might be able to expose some exception checking function from libc. https://github.com/llvm/llvm-project/pull/199808
    - Joshua: I'm working my way through exception behavior. AST interpreter is not catching much cases, bytecode is better. Found couple GCC bugs.
    - Tue: I might be able to refactor them into libc functions
    - Joshua: the problem is that the exception behavior is not correctly hooked up in the interpreter
    - Hubert: maybe we're talking about the wrong PR
    - Joshua; I've been talking about the large PR
    -
- Support for Block Pointers as Non-Type Template Parameters (RFC)

Meeting Agenda - May 20th, 2026

- Clang 22.1.6 is out
    - 22.1.7 is scheduled for June 2nd
    - 23.x will branch on July 14th
- For Awareness:
    - Open Access to ISO Documents (RFC, draft testimonial)
        - Aaron: if you're working for a company which would like to submit a testimonial, I'm willing to help to deliver it. I expect downstreams to have slightly different arguments: your users might need access to the documents
    - Static analysis working group will meet on June 2nd (post)
    - Reid published an RFC to switch from reST to MyST markdown
        - Not just a policy change, but also mechanical conversion of several key docs (template, langref, cmake) and encouragement for more PRs with mechanical conversions
    - Update the semantics of the noescape attribute (updated RFC)
        - Aaron: updated after CAT meeting
    - Removing Templight support (RFC)
        - No maintainer, very painful to update the tests
        - Aaron: unclear if downstreams use it
    - On the forums, Nikita published an RFC about strict FP
- Hubert: stdatomic.h
    - Hubert: there was an item from the last item that we're waiting news on
    - Vlad: I intended to be present for the libc++ meeting but I missed it. Did anyone make it? I checked out the notes, but Louis didn't make it either. So I don't think this got discussed.
    - Corentin: should we set up a meeting with libc++?
    - James: what was the previous action item?
    - Hubert: to discuss on libc++ to see if they had further design ideas and whether we agreed with the approach.
    - Aaron: it would make sense to have a meeting. Do you want CAT to do that?
    - Corentin: yes
    - **Corentin takes an action item**
- Hubert: `-finput-charset`
    - Hubert: FYI we're making progress on putting it together to publish a PR, ETA early June. PR is going to be much smaller than the -fexec-charset, so it might make it into the release. I hope Corentin can keep an eye on that
    - Corentin: I'll try, but keep in mind we have WG21 meeting in Brno coming
    - Hubert: who might also help with the review aside from me and Corentin? The problem is I wrote a lot of code there myself.
    - Aaron: Tom Honnermann stepped down as a maintainer, but he has interest in this area. I might also help

    - Hubert: this won't be as domain-specific, unlike -fexec-charset. We'll need to reorganize the PRs to split out the common parts
    - Aaron: for example, I don't think we need to care about clangd changing things in the middle
- Hubert: constexpr cmath (PR)
    - Hubert: PR is up, there is minor activity. I have some comments. IIRC this is the fourth attempts over the past years. The scope is nice, it avoids hard questions. I hope author will continue with it. I'm concerned about more involved changes. The first attempt didn't go forwards, because people didn't have time to think about this space. The second attempts went down because author decided to prioritize other PRs. The third attempt was successful in a sense that we came to a consensus how it should interact with fenv_access. This fourth attempt doens't need to be stuck.
    - James: can you link previous three attempts in the PR?
    - Hubert: I provided a link to the third attempt. I'm not sure how useful are the first two. We don't necessarily need to bring past uncertainties
    - Corentin: we're adding all new C functions to C++, so we're going to have more. It's a lot to implement
    - Aaron: I thought we're going to do hand-in-hand: libc implements them, then we reuse them.
    - Joshua: constexpr stuff is being added is the same as sine routines. There's not that much of a difference. We have a framework
    - Aaron: so this is a preparatory work?
    - Joshua: for C and C++ stuff. More work will need to be done, though
    - Hubert: I'm happy to see this patch continue. This will be a precedent that future functions will follow. Size of the patch looks large, but it's for its benefit. Right now there is a lot of problems with negative zero values in the current patch
    - Joshua: I need to check how exceptions are handled
    - Aaron: yes, this is what I also saw
    - Hubert: yes, special cases are worth looking into, but negative zeroes do not always involve special cases.
    - Shafik: author seems to be responsive so far
    - Aaron: we should asks for more tests for edge cases: negative zeros, etc. Or this is not beneficial?
    - Shafik: we've been discussing whether to do it in this PR or in the next PR. I think we need to do it, if it's the next PR, it's the very next PR
    - Aaron: I agree
    - Corentin: I'm assuming libc has a lot of tests for this. Can we reuse them?
    - Aaron: good question
    - Joshua: edge cases were tested locally, even in the absence of tests
    - Hubert: this patch doesn't exist in isolation: it's calling other code to get results. This is white/grey/black box question: we need to test parts where we interact with the existing code. What this patch is actually doing, and what I'm interested in, is testing when constant evaluation should fail

- Aaron: this matches my concern. We want to diagnose UB and fail the consteval. Denormals, etc. Whether it needs to be exhaustive is another question, but we need to decide upfront, because other work will follow
- James: it seems that we shouldn't be implementing any math. APFloat is supposed to do the work, like exception flags, and what it returns should be tested. Code here should be straightforward: call APFloat, check exceptions.
- Aaron: do any fp exceptions make anything not a constant expression
- Joshua: inexact doesn't make it a non-constant
- Hubert: nexttoward has interesting behavior around underflow
- Joshua: there is also a paper by Jan that takes regular fp arithmetic and replaces exceptions. There anything except for inexact makes exprs non-constant
- Hubert: fp exceptions will cause non-constant expressions. There are functions that write via pointers. Even if we get math API properly, constexpr machinery will have to be correct. I don't have much experience with the bytecode interpreter, but lifetime-wise, if you have a pointer and you write through it, it seems that calls being made are wrong. I made suggestions on the tests to remove unnecessary initialization, because it exposes crashes.
- Joshua: the biggest problem of some functions is that they are not defined by IEEE 754, they have C definitions. APFloat doesn't support them
- James: let's implement them
- Ariel: I'm also concerned with APFloat. I'd like to note that APFloat can have non-ieee representations. Whatever we do to the APIs, we need to be mindful that it's not just non-ieee format
- Aaron: I was also wondering what's going on with double double
- Hubert: what's happening there is that it was never a default abi on AIX. Forlinux on ppc, default long double is transitioning on many distribution to ieee. Work on this in clang and llvm has not been completed, but people are supposed to be moving away from it.
- Joshua: in terms of non-ieee representations: x87 pseudo-nans, -inifities are incorrectly represented in the first place
- Aaron: do we have machine mode flags that are not modeled by APFloat. What happened if user disables SSE2?
- Joshua: denormal flushing is a can of worms we don't model. You'll get a different env between compile and runtime
- Corentin: this is the only sane thing to do
- Joshua: same env is only recommended
- Aaron: what the users should expect?
- Vlad: ABI breaks
- Shafik: does anyone have examples? Is this theoretical?
- Joshua: typical example that show differences in denormal flushing. I have an example that gives different results on different hardware
- Aaron: I've seen them, too. Standards give us some wiggle room to return different results, but that's not what users expect
- Corentin: GCC has implemented constexpr cmath long ago

- Corentin: we'll be able to get constant folding for free. Users seems content with it
- James: those problems seems more benign in consteval than in optimizations. In runtime it leads to miscompiles, but no one can tell the difference from UB. But rust cares about soundness. Optimizations need more work. For dynamic rounding modes we can say that you get the default rounding modes. I'm not that concerned about this set of errors for consteval
- Joshua: biggest issue with miscompiles are sin and cos giving different values. Users will notice and complain. When it comes to the cases in this patch, they are all outside of denormal mode, so they are exact.
- Hubert: I wanted to mention that for GCC side this is was on the issues that clogged us up. GCC made non-prefixed versions constexpr. THere is prohibition in the standard library to do that, but when I told GCC they said they won't fix it. This might be reasonable for us to adopt. I wasn't aware of other additional modes that you were talking about, but at least for rounding mode, or with fenv_access, we did come up with how to deal with there in the review prior to this one. It was communicated to the author of the current patch. In the frontend we also try to do some folding. We will not do folding if fenv_access is on. This is an open topic in the review. I'm leaving it to the author's discretion what to do with rounding modes for functions where they do not make any difference.
- Aaron: I agree with this with the caveat that we're not doing this in C?
- Joshua: there is an annex saying what has to be evaluated at compile time
- Matheus: I'm not doing much in fp, I have a bunch of PRs that wait for review. About equivalence of declarations
- Aaron: so it's a different topic
- Ariel: what is miscompile?
- James: the code that the user wrote is correct, but the generated code doesn't do what it was supposed to do
- Ariel: how is this different from compiler error?
- Matheus: miscompiles are silent
- Shafik: I think some differences between compile and runtime env is unavoidable, this is an educational question. I'm happy to write blogposts if there is someone who can feed me examples.
- Joshua (in chat): I brought up some of these issues in WG21, their response was essentially "this is already a mess, no issues in making it even worse then"
- Aaron: specific to this PR, we have an ideal that we're striving for (same env), and we should keep this position.
- Shafik: but I'm hearing differences are unavoidable
- Aaron: yes, but what is our default position?
- Shafik: maybe I misunderstood, but I'm hearing that it's a matter of a poison to pick
- Aaron: I'm talking about the PRs in question. Do they need to come with a justification why they need to diverge?
- Shafik: unless author is an ieee expert, it should be on reviewers

    - Vlad: as long as it's documented, the reviewers can explain to the author why the patch has to pick the behavior they did. We need the explanation for Shafik to have as background for his blog post
    - Hubert: I wanted to say that in general we know that it's not reasonable to expect envs to be the same, because we're not going to emulate every single target in terms of the algorithm used to implement the function. Current patch is doens't really touch those problematic functions. I don't understand why people feel there is something to say about this
    - Aaron: C insists that envs are the same. WG21 is more relaxed. I think WG21 is more reasonable. The reason is that users have this expectation. What I hear is that we're make them compatible as much as we can, then deal with the bug reports
    - James: this is not new, we're doing all this in optimizations. What's new in consteval is interaction with dynamic environment. Functions specified to be constexpr have to be, we can't go around it. Expectation on the C side was that evaluation happens in default modes, calls happen in default modes, unless it's fixed rounding mode, which is special and exists for special macros and doesn't translate to C++, because they don't exist in C++.
    - Aaron: it's exacerbated in C++ now that you can you use FP values as NTTPs.
    - Corentin: same compiler will have the same results for the same NTTP, because of ODR. If users are happy with C++23 mode, we can backport to other modes
    - Aaron: I think C expectation is because of file-scope initializers
    - Joshua: the rules are that some things have to be evaluated as if at compile time
    - Corentin: what GCC does?
      Aaron: I don't know.
    - Corentin: we should check
    - Aaron: agreed
    - Corentin: our concerns might be theoretical, because people don't complain too much there

- Did not get to the following topics:
    - Requiring C++20 Modules Serialization Tests for New Features (RFC)
        - Aaron asked Chuanqi to write documentation about how to write serialization tests
    - Builtin names for C++26 atomic reductions (P3111) (RFC)
        - Vlad left a comment wondering whether `__builtin` prefix should be used
        - Louis and Aaron want the prefix, Jonathan Wakely doesn't
    - GCC2 ABI (RFC)
        - Haiku wants upstream support for GCC 2.95 ABI for the features that compiler supports
        - Scoped to codegen; no semantic analysis changes

        - James suggested to keep this out of the compiler entirely: provide a shim library instead, so that old applications that need GCC2 ABI can interface with newer system binaries that use modern ABI
    - Support for Block Pointers as Non-Type Template Parameters (RFC)

Meeting Agenda - May 6th, 2026

- Clang 22.1.5 is out
    - 23.x branch date still expected in mid Jul
    - Corentin: I don't think there is anything major
    - Ambrose: if you'll review all expansion statements, we should be good to get them in 23
    - Vlad: we should be landing this sometime in June. Merging it before the release cut-off is not good
    - Corentin and Ambrose: yeah
    - Corentin: Hubert, do you need EBCDIC in the next release
    - Hubert: I don't have an answer right now. We're still working on the -finput-charset. EBCDIC is not the only thing we're intereseted in, we also need Shift-JIS. Ideally we'd like to have in 23, but we understand it might not happen
    - Corentin: I'll try to find time for that
    - Hubert: for the -fexec-charset, I believe we have issues that haven't been addressed. We're working around them downstream.
    - Corentin: can you keep the non-encoded source string?
    - Hubert: my guess is that it works for some things but not others. We need to know where the string is used. I think we have a patch stuck somewhere.
    - Corentin: one solution is to disable checking if source encoding is not UTF-8
    - Hubert: yeah, I think we've ended up with that. But if the user declared their own format function, it's not covered.
    - Corentin: if we disable built-in, are there side-effects?
    - Hubert: one is format string checking for diagsnotcis, the other one is similar, but for transformations.
    - Corentin: is codegen affected?
    - Hubert: I'm not aware of many cases when we do the transformation. %s to puts, for example. You're calling an i/o function, and it's unavoidable that you call it.
- Static analysis working group is being formed
    - https://discourse.llvm.org/t/rfc-forming-a-static-analysis-working-group-in-the-clang-ecosystem/90719
- Nicolas: _Atomic and <stdatomic.h> in C++ language modes
    - https://github.com/llvm/llvm-project/pull/143036
    - https://sourcegraph.com/search?q=context:global+lang:C%2B%2B+_Atomic+file:.*%5C.cpp+-file:.*test.*+count:1000000&patternType=regexp&case=yes&sm=0
    - Nick: there was an issue with _Atomic in C11, to make it compatible with std::atomic. C++23 made them ODR equivalent. The question is what do we do with the old code. Project use C and C++ headers, and see confusing error messages. Android patch makes atomic compatible in all modes.
    - Nicolas: the main problem is _Atomic in C++, which clang has been supporting for a long time. C++ header stdatomic.h defines _Atomic to be std::atomic. Would be nice to find a solution to tell the users what is going on.
    - Vlad: were _Atomic and std::atomic different?

- Nicolas: layout would be the same
- Hubert: you can't pass it by value in many ABIs
- Nicolas: I think we should align how this is passed. No one really relies on the ABI. The bigger issue is depending on the standard mode in, _Atomic might be an extension or std::atomic, also depending on the macro
- Hubert: libstdc++ and libc++ also do not agree on the layout of the type.
- Corenting: precumably, libstdc++ cares about C. should the layout be in the itanium abi?
- Nicolas: platform abi decides what the layout of _Atomic is
- Corentin: I found the C++23 paper: https://www.open-std.org/jtc1/sc22/wg21/docs/papers/2020/p0943r6.html. The authors knew we have a concern, but I don't think they understood it. Should be go back to the WG21?
- Nicolas: I thought about it
- Nick: in the libc++ discussion it was pointed out that gcc doesn't support _Atomic in C++ mode because of the different frontends. Would they care?
- Nicolas: yes, they should have the same concern.
- Nicolas: I'd like to know how to fix the API. I'm not a person to talk about ABI
- Hubert: at the end of the issue there is idea that _Atomic disappears if the header is not included. A keyword will be added to get the plain _Atomic even if the header is included. I think it is sensible. I don't know if this direction helps android, though.
- Nick: Aaron's feedback is that clang doesn't unship extensions.
- Nicolas: without the header you'd have a warning that ABI will be incompatible
- Hubert: I think we have an idea that clang will continue having the extensions, but I don't think we know what libc++ will do.
- Nicolas: I think that in order to have some consistency we'll provide stdatomic.h in all modes. This will break some users and the clang extension as-is
- Hubert: because everyone's programs who want a C header gets libc++'s header.
- Nicolas: yes
- Hubert: can't libc++ include_next the C header in old modes?
- Nicolas: not really, I think. I think we could, that's the status quo. But there's a code that uses _Atomic, and depending on the presence of the header you get different codes.
- Hubert: when people switch language modes they expect changes more than when they change versions of the compiler
- Vlad: I think this is what the committee wanted.
- Hubert: it's hard to talk about the committee, because it's a room full of people. The header didn't exist in the world before, and now it's not clear if it exists in the old modes.
- Corentin: from clang perspective if we're providing _Atomic, we might need to supply our own header. If we don't do that, we'll never be consistent

- Hubert: but the extension is to use atomic without C++ header. I'm seeing a world when you get C header in C++ mode and get the same thing in old language modes
- Hubert: if you want to not include the header but provide _Atomic, then you're saying you know what std::atomic is without seeing the definition.
- Hubert: libc++ header might define a macro to jump to C header, and we can document it.
- Nicolas: I don't think it's a great idea
- Nick: to restate the suggestion: users can use _Atomic without headers, but if it's redefined by the headers, we diagnose?
- Nick: I'm trying to understand how extension works in the presence of headers
- Hubert: it's a macro
- Nick: the idea is to see whether we can diagnose it?
- Corentin: I think it's a good a solution
- Nick: this will fix the problem of using keyword without the extension
- Hubert: but we still have a problem if you include the header, and you got C++ header instead of C header
- Nick: say we add the diagnostics. How does it help users in per-C++23 modes?
- Hubert: we can enable it by default, but it addresses only one of the concerns. I think libc++ should warn users about _Atomic in older modes.
- Nicolas: I think it would be reasonable to mark the macro as an extension in pre-C++23, given that clang provides the ability to do that
- Hubert: why do we want to mark the macro as an extension instead of the inclusion of the header
- Nicolas: we also can't mark headers as deprecated
- Nicolas: I tried to use #warning, but I broke the world. I think it doesn't respect system headers, but maybe it can work
- Nicolas: #warning is emitted from the system headers
- Vlad: do we know what to do?
- Corentin: we don't have a solution for ABI
- Corentin: _Atomic used to decay, but not anymore
- Corentin: in C it becomes an int if passed by values
- Nicolas: but it defeats the purpose of atomics
- Hubert: you can take an address and pass. All the arguments that attacked the paper that deprecated volatile apply here
- Arial: deprecated everywhere?
  Hubert: only where it is questionable.
- Corentin: do we say we only warn?
- Nicolas: that's the only thing we can do without breaking anyone
- Hubert: on by default
- Hubert: people might want to keep status quo, because that's what they had
- Nicolas: the only reason is ABI concerns
- Hubert: again, you can pass pointers in, and it's in the mangling for C++ linkage

- Nicolas: we can provide overloads to allow the extension, we don't need the C header
- Hubert: i believe that makes sense. You can make them C linkage, and then it will become C header
- Hubert: we might tell users to undefine the macro
- Nicolas: it's not a beautiful solution, but it is a solution
- Nick: #warn is all-or-nothing in command line options.
- Hubert: about the #warn: this is another not beautiful solution, but we can ask to define a macro to suppress the warning
- Nicolas: yes, we can do that
- Nick: is there prior art?
- Hubert: yes
- Nicolas: that is acceptable. We did it with nodiscard
- Nick: at that point it's a #warn in libc++ headers, and no diagnostics in clang?
- Nicolas: I don't know. We want to warn the users of _Atomic extension
- Nick: so we already have -Wc11-extensions
- Hubert: we want to enable it by default and massage the diagnostic group
- Hubert: I'd like to take it back whether the diagnostics exists. We want to say that it's a conflicting extension. Probably a new diagnostic?
- Nick: add notes?
- Hubert: for the API there might be progress until the next meeting. For now we have PRs. For ABI we can't make too much progress offline.

Meeting Agenda - April 15th, 2026

- Clang 22.1.3 is out
    - 22.1.4 expected to release next Tue (Apr 21)
    - 23.x branch date still expected in mid Jul
    - Aaron: I didn't see major issues coming in. .4 seems stable
    - Shafik: I haven't seen anything concerning
    - Hubert: this reminds me about an issue we found downstream, looking if it's reproducible upstream. Can't bootstrap because of MLIR.
    - Aaron: I wonder what kind of testing infra we're missing. Or is this specific to your downstream?
    - Hubert: someone else hit by that. They tried to work around in MLIR sources, but abandoned the patch.
- Vlad: D4138R2 CWG3103 "Corresponding members and by-value object parameters"
    - Vlad: looking for feedback from our implementation, do not expect to have a uniform position on this (Corentin has concerns with the proposal), but maybe we can agree whether we want to issue some diagnostics by default for some of these examples regardless of how the DR is resolved.
    - Vlad: <background description of the DR, presentation of the paper>
    - Corenin: I have opinions on the design, but I think that's more a question for EWG. I think we can implement anything that EWG comes up with. I'm opposed to doing anything until we get a reply from evolution. I don't think anything is on fire; writing these overload sets is pretty rare.
    - Vlad: If we have an opinion, we should know. FWIW, people do write this, at least one person said they need these overloads (Gasper), but maybe I'm missing something because I don't understand why. It has something to do with move-only types. I'll ask him again.
    - Corentin: there is no reason to make that code ill-formed; because they are different types. We shouldn't make it ill-formed just because it's not useful.
    - Vlad: should we diagnose those by default?
    - Corentin: no! I disagree with that. They're different types. That's why I want to wait for EWG.
    - Vlad: but users would benefit from finding out up front that the declarations may be useless instead of waiting for them to try to call them as an overload set.
    - Corentin: for Clang, if you come up with a warning that warns on this and not just that combination, that could be a good discussion. But just saying we don't want to allow this thing but not other, equivalent scenarios, doesn't make sense. That's not consistent.
    - Aaron: I do think this kind of discussion is good for this group because we meet to discuss what our community position is (if we have one) and to gather feedback to help inform EWG. Our job is to inform the committee, not to wait for the committee to decide what our QoI should be.
    - Corentin: no implementer opinion from me, we can implement whatever.

    - Hubert: I'm slightly lost here. We have the paper, it proposes to make certain things ill-formed at the time of declaration, are we saying we received some sort of feedback (maybe the committee didn't hear) where these overload sets may exist but the user doesn't care that the operations are usable or not?
    - Corentin: My argument is that making it ill-formed is inconsistent in a language design point of view. If you have a valid justification, what is it?
    - Hubert: The justification (in the abstract) is that you're probably expecting people to be able to call it, and if it's statically determinable that it's not callable, you likely want a diagnostic. But maybe we just want to consider a QoI diagnostic and that doesn't stop users from being able to have these, so we can gather feedback. So adding the diagnostics now in Clang is useful for gathering this information if we're willing to take the risk of adding the diagnostics. But there's already reasoning that this paper might go through, but we could gather information about the impact. If it goes bad, then we've learned something.
    - Corentin: https://compiler-explorer.com/z/veezf4Er4 – if we want a diagnostic, why are we not warning on that because it's the exact same case?
    - Hubert: I think that the standard kind of thinks that unbounded overload sets are slightly different from class scope ones.
    - Shafik: I think I agree, those are fundamentally different in some ways. Not apples-to-apples.
    - Aaron: does = delete play into this so people make an overload set and delete one specifically?
    - Vlad: I don't think so; you either never get the delete message or you always get it in this case.
    - Aaron: specific to whether we should diagnose, I think we *should* diagnose by default. The overload set is nigh-useless because you cannot call the functions without contortions. Why wait for the librarian to discover that via their users when we could warn them up front they wrote something very suspicious?
    - Vlad: so far I've been focused on use cases outside of templates, do you have a hunch if there's something involving templates that might change anything here?
    - Hubert: it would help to have more people from libc++ involved for answering that
    - Vlad: good idea, I'll do that.
    - Aaron: so do we have a community position on this? I don't think we have one, but we might consider implementing the warnings to explore more.
    - Shafik: I don't think we have a community position
    - Hubert: agreed, we want the more information from libc++ folks
- C implementation effort update
    - Aaron is continuing to reach out to downstreams. Have reached out to: Intel, IBM, Apple, NVidia, AMD, ARM, Qualcomm, Sony, and Google; am waiting to hear back from a few. Thinking of reaching out to others, but other organizations have historically not been working on standard C support in Clang so it's unlikely to move the needle much on getting more implementation effort.
    - Responses received so far suggest this risk will be an ongoing concern.

- Starting to pull together more data for a presentation with the expectation that this may need a Clang Area Team-hosted meeting to discuss next steps.
- Aaron: If you know a downstream C compiler not on the list, let me know.
- Aaron: C support has been winding down over the years
- Aaron: if it sounds like no one is putting in the work, we'll have a CAT meeting about this.
- Aaron: downstreams usually inherit standards support, and not implement features downstream. If no one works on it, that's feedback to WG14
- Aaron: downstreams might be interested in knowing the answers, because their users might start asking for new C features
- Corentin: we might have the same situation in C++
- Aaron: not wrong, but I see an order of magnitude more people there
- Aaron: situation is more dire for C. the work that happens is only occasional
- Aaron: number of companies said they're frustrated with C
- Shafik: I thought the lack of resources is because of something different, not frustration
- Aaron: companies saying they are frustrated about ignored implementation concerns
- Aaron: companies are saying WG14 outpace what users are asking for
- Corentin: I'm looking at C status page. Do you have a list of big features which are not implemented? It seems we're in a good shape
- Aaron: yes and no. This is also about pace. On DR status page we didn't do anything for 2.5 years. So it's not like "reflection is huge", it's committee adding things no one needs
- Shafik: maybe we should put news out there that we deprioritize C work to make users complain about things they care about.
- Aaron: we might go in direction where we don't by default implement what wg14 adopts. Maybe we'll do this for C++
- Shafik: for C++ it might not be needed
- Hubert: Trying to keep this within the context of ISO/IEC standardization: Is there a misalignment between the evolution of the C standard and market interest?
- Aaron: I think so
- Hubert: can we formulate this feedback as a CD ballot response?
- Aaron: good question. I thought of inviting WG14 convener to the aforementioned meeting with microphone unplugged. Not necessarily formal, just that might be enough
- Vlad:
- Corentin: we should keep in mind there is a lot of latency in user demand. I think there's no lack of interest in new features.
- Vlad: in the collected data, unlike C++, C users are predominately using C99. If a new feature is specified in a way to make it impossible to support, users will please say "do something to backport the feature". C users seem to be even further behind in terms of adoption.

- Aaron: yeah, it might be the latency, but then there are features no one asked for them. For instance, when we disallowed bitint as an underlying type of enum, we wanted to avoid promotion behavior. When sdcc wasn't happy about the restriction, this was adopted as a feature. So the way this feature came to be is questionable. Part of the concern is that C used to standardize existing practice. We did that occasionally. Today it's an everyday occurrence. If something is in the standard, it's expected to get implemented at some point. Even if we get one company to care, they can disappear. This needs to be a community effort.
- Aaron: we need multiple companies
- Corentin: companies expect someone else to do that.
- Aaron: I'm socializing this, because this is a community issue.
- Aaron: C side should be an easier problem to solve than C++, and that might smooth things for C++
- Michael: I've heard ramblings about implementers needing time to catch up with C++26. Was there any discussion of that?
- Aaron: yes, a paper was written and presented. Committee is aware of this, but no action items came out of that. SG was proposed. Another proposal was to make EWG and CWG meet together.
- Corentin: this is an accurate summary. There was no firm commitment to do anything. I don't think
- Vlad: have you heard any news about Louis' paper about C++29 being a bug fix release.
- Shafik: not heard any news yet, but I've poked him about it. He wasn't in Croydon. I assume he's buried in work at the moment.
- Hubert: Michael, if you're interested, we did a review of WG21 meeting in this meeting previously.
- Aaron: it's not like things are not of fire, but we will move slowly. Need to come up with the next steps, though. If you know downstreams who needs C implementation, let me know.
- Michael: There is google usage of clang build of linux.
- Aaron: I reached out to google, because they've been driving libc implementation. But they predominantly interested in C library, not the language.
- Hubert: I suggest that for this specific topic of C implementation, if it comes up at the next meeting, we scope it to new information.

Meeting Agenda - April 1st, 2026

- Clang 22.1.2 is out
    - 22.1.3 expected on Apr 7, in a ~week
    - Aaron: I haven't seen anything major, but we have some stuff on the plate for discussion today. Sometimes we make changes that we're kinda certain are in a good shape, but in 22 there were two issues that we should've caught before the release. So there is a question of how to do that in the future.
    - Shafik: I didn't see anything worth mentioning for 22 release
    - Hubert: I opened an issue 188640 about subsumption. My understanding is that it happened because of a lot of work. We'll fix forward, not sure if we can backport it.
    - Aaron: if it's going to be ABI-breaking, backporting is less likely. We've seen a number of things we want to backport that would break ABI. If we see something worthwhile, 22.2 will happen
    - Hubert: I didn't knew it's possible
    - Aaron: I'm not sure it ever happened
    - Vlad: I think it happened for clang 7; I've seen it on CE
- WG21 met in Croydon last week
    - Aaron: WG21 finalized C++26. No major surprises. Reflection is still in C++26, so are contracts.
    - Shafik: the most controversial things were two papers competing to fix CWG3150, where two groups of P2996 authors were competing. I've heard from authors that they wanted more time to explore the design space. So I thought maybe we don't need to fix it in C++26, but it was a tie in Core. Then there was a poll on the approach, and the one easier to understand won. Then there was EWG discussion which went off the rails for at least one presentation. In the end EWG decided to not fix it in C++26, but status quo is known to be broken. Then there was an issue about constification in splice-expressions.
    - Hubert: On the topic of consteval-only, I understand why there was a presentation that wasn't well-received. Types approach is the status quo. Until the last meeting consteval-only types were the way. That said, values could get us better results. So deferral was the right decision. Values might not be a slam dunk
    - Shafik: I'm sorry if it came out like this. It's just value-based approach looked much more promising during the meeting
    - Hubert: I made a joke during the meeting that consteval should've been "constexpr!"
    - Aaron: there was an evening session about implementation reality of WG21 standardization (paper link?). As someone who was calling in, I'm not sure that this meeting had the intended effect. Some of the paper authors weren't in the room. I don't have impression that WG21 will make changes, but I'm under impression that they will try. I don't think implementers agree that this will solve

the problem. Other changes are needed to reduce the throughput, e.g. combining some of the working groups.

- Shafik: person who was supposed to to present the paper dropped out in the last minute. Erich tried to step in, but there was a miscommunication, so paper wasn't presented. Being in the room, I felt strong support for more joint sessions between EWG and CWG. Not sure about anything else
- Hubert: When we do those big meetings, library has to be considered. Language and library have difference experiences, even if they solve similar problems. LEWGI was amenable to making changes, e.g. asking for all the relevant examples for the presented features. Comprehensive testing aspect is something we want to keep drumming the beat on. I wouldn't be as pessimistic, because committee can ask questions that would make it more implementable. The problem is that some authors are far away from implementations, and it's not clear how to make process work for them.
- Aaron: I agree with a lot of the points you raised. On the language side of things, I've heard the same thing as during previous ones: asking for implementation is too much for an average paper author. It would also favor people who have minions to implement what is proposed. I continue to be worried about implementations in a fork of clang being presented as an implementation in Clang. We need to push more on deployment experience. "I hacked this together ignoring extensions, etc." We have users who come and ask "why is this not in clang when the paper said it's implemented in clang?"
- Shafik: I think people weren't convinced that there is an issue. Piecemeal presentation wasn't helping. People were left with an impression that there are issues, but with no general understanding
- Hubert: I was under a different impression. People understood the problem, but solution wasn't clear. Presentation was scoped on specific problems. It was done in a way that assumed that people have read the paper. It was focusing on solutions instead of on the problems. I haven't heard anyone saying that they don't see the problem, though
- Aaron: wg21 folks, do you think clang wg can do something that makes us happier?
- Hubert: who here sits in EWG?
- Shafik: Erich is chair
- Hubert: he has to wear chair's hat, too
- Vlad: in Sofia (maybe), one of Davis' papers was presented to EWG, it was a great move to ask the room "who actually understands what is being presented?". When almost nobody raised their hands, EWG decided not to take polls. That is a useful question to ask more often, and it wouldn't interfere with his chair hat. But he's not here to say otherwise, either.
- Aaron: we don't have anyone sitting in EWG
- Vlad: I've seen Corentin there often. I myself spent more time there, but I'm not a fast thinker to stop papers on the fly

    - Aaron: I've spent most of the time in CWG, but if I could clone myself, the clone would go to EWG
    - Hubert: when asked about implementation concerns, I informally replied that this is mostly CWG's concerns. CWG loses sight what happened during the cycle. LWG, who are affected, are the ones who remember. Maybe we need LWG send someone to EWG
    - Vlad: library working group specifically has the least number of members of all the groups (potential quorum issues for sparing people). Core losing sight of changes may be partially because some of the examples we wanted to put in the standard were incorrect. It might be indicative that the folks who should have the most accurate view of the standard are swamped by the changes and so those changes don't sink in. aka, an indication we need to slow down
    - Aaron: anything else? I don't want to turn this into WG21 meeting. WG21 folks: reach out to me if you have ideas. We can have 1-on-1 if needed.
- "16% compile-time slowdown after clang (#161671)"
    - Aaron: how do we avoid this kind of slowdowns in the future? Sometime we do speculative changes, and wait until RC
    - https://wiki.debian.org/qa.debian.org/MassRebuilds
    - Aaron: I haven't look into it, but it seems it's a corpus we can compile to see what the fallout is. Not a silver bullet, because C23 and C++23 code is probably non-existent there. But this can help.
    - Aaron: specific to this particular issue, we introduced the regression, and we need to fix it, but Corentin is not here to speak to it.
    - Vlad: this issue was filed in the middle of Dec and it likely didn't get enough attention due to the holidays. The situation is not good though.
    - Nick: can we get a C++20 workload on compile-time-tracker?
    - Vlad: a bit worrying, his hardware is likely already taxed by the current workloads, adding more is asking a lot.
    - https://discourse.llvm.org/t/the-need-for-better-frontend-benchmarks/90257/
    - Aaron: I've seen a thread about this (^). A trick is to find the workload, and then being able to set custom flags. Generally we're in favor to find a solution here, but who is doing the work to maintain it?
    - Aaron: Corentin got back from WG21, so he might have more time for this
    - Vlad: not necessarily, given his life circumstances
- C implementation efforts
    - Much heavier community focus on C++ than C, but C resources seem to be drying up over time
    - Concerns about lack of attention to C causing long-term problems for C23 and C2y support, how do we get more folks involved in implementing C?
    - Looking for additional C Conformance maintainer in Clang
    - Aaron: it's not easy to tell who is working on what, but it's much easier to understand who's working on standards. It's no surprise that C++ has more support, but C resources are drying up. We used to have 2-3, now we have less than one combined. I'm worried about the future of C in Clang. I've been asking

downstreams for support. Along the similar vein, I personally hope to step away from WG14 after C2Y is released, which is 3-4 years away. I'm searching for someone willing to be C conformance maintainer, and preparing them to join WG14. Reach out if you know someone who could fit.
    - Hubert: as an open source implementation, the actual problem is when the committee puts something in that we're not interested in implementing but people come with patches to implement it, but there's no way to get those past the bar for inclusion.
    - Aaron: that puts us between a rock and a hard place, because feature is in the standard
    - Hubert: it will still eat up review resources if someone comes with patches.
- Vlad: D4138R2 CWG3103 "Corresponding members and by-value object parameters"
- Project council meeting immediately following this one

Meeting Agenda - March 18th, 2026

- Clang 22.1.1 is out
    - Mariya: how do you decide whether to backport a change or not?
    - Aaron: bug fixes, no ABI changes (to user code or compiler), non-risky changes; ask for opinions when in doubt
- WG14 meeting
- Upcoming WG21 meeting
- Increasing number of “pure-AI” contributions from new contributors. Can we somehow stop wasting time on these? Maybe not allow new contributors use LLMs similar to not allowing using LLMs for good first issues
    - No assisted by label on these
    - Content is often not relevant
    - PR description is too long and does have Markdown that no human uses for commit messages
        - Sometimes they quote obsolete standards in a very unhelpful way (example)
    - https://discourse.llvm.org/t/concerns-about-low-quality-prs-being-merged-into-main/89748
    - Example https://github.com/llvm/llvm-project/pull/181579
    - Aaron gave a brief overview
    - Mariya: I see a lot of PRs where people didn’t spend a lot of time on, just shoveled us the LLM output. I’m glad I’m not alone seeing this and being concerned by this. This is not limited to Clang, but I brought it to see if others were seeing issues. Despite all this I’m not in favor of no ai at all; I would say it can be used, am also under pressure from my employer to use AI more. I think these tools can be used if the user has a general idea of what they’re doing to help understand other areas of the compiler. e.g. doing SYCL stuff in the community, I should not have to be a maintainer to use the tool there. Maybe limit it to just disallowing use by new contributors
    - Hubert: I kind of agree with that. This is a somewhat classic conflict of what the community wants; we want newcomers, but they use AI in ways that they frown at.
    - Aaron: I hear that and agree somewhat, but this also gets used incorrectly by experts
    - Vlad: the fallback idea makes for tiers of contributors, we’ve historically avoided that. When is project council meeting next?
    - Aaron: project council is scheduling currently, expected Apr 1 will post more when we know
    - Hubert: people make the same mistakes in terms of “wrong fix at the wrong level”, it’s not just an AI thing, it’s an experience level thing
    - Aaron: it’s a scale issue; we get way more PRs than we did before
    - Vlad: if we go with no ai, do we expect to get less PRs
    - Aaron: it gives cover to just close the PRs

    - Vlad: how to detect that, there aren't really good heuristics. What's the end game?
    - Aaron: my goal is "pause until the tools exist to help with exactly that."
    - Hubert: maybe multiple cases here?
    - Vlad: how many options do we really have now that it's established that AI output is not copyrightable?
    - Aaron: unclear whether this applies to code (or just images) it likely requires foundation consideration.
    - James: AI tooling in general is useful; compilers are extremely complicated and not well-documented, so both humans and AIs make mistakes. If a human does the work, they're more likely to stay involved to make it work because they sunk the costs. It's so much easier now with tooling that unmotivated people can do it. The tools aren't going to learn and the folks may not stick with the community.
    - Aaron: definitely seen some of this happening
    - Hubert: part of the friction is if the person doing the review is interested in the area, they may be more willing to work with the tool output even if they wouldn't otherwise, which can feel arbitrary to new contributors.
- On the forums
    - Markdown support in ASTComment (RFC)
    - null_after_move PR is up (PR, post)

Meeting Agenda - March 4th, 2026

Agenda:

- Next meeting will be an hour earlier for people in ~~blessed~~ non-DST timezones
    - Aaron: next meeting is March 18th, check the time
    - Shafik: I won't be there, because I'll arrive early in Croydon
- Clang 22.1 is out
    - Aaron: if you were doing work for 22, now is the good time to check bug tracker. Triagers do a good job bisecting the issues, but help is always wanted there
- Upcoming WG14 meeting
    - Agenda: https://docs.google.com/document/d/1v3jKa9lxtsHL0jOQ-3uQmVcjG7Gz6-2aoLSGd5u-R3I/edit?tab=t.0
    - Yeoul's "dependent attributes" are not on the agenda, neither are forward declarations of function parameters
    - Krause, Named Address Space Type Qualifiers for C2Y v3 [N3795]
    - Colomar, Adopt qualifier conversion rules from C++ [N3749]
    - Aaron: WG14 meeting next week
    - Aaron: we've had discussions about forward declarations of function parameters, but they won't be discussed next week
    - Aaron: named address space qualifiers are on the agenda. We have some of that in form of __global and addressspace. Author tries to bring TR from 2003 straight into the standard. It;s not clear if anyone actually implemented the TR. We just started documenting it, not clear so far if TR is implemented. TR was written in the time of segmented architectures, but GPUs are not mentioned anywhere
    - Joshua: opencl is mentioned twice, but I'm not sure opencl actually does this
    - Aaron: if you have feedback, reach out to me. WG14 attendance is not required
    - Ariel: how does it correlate with the similar notion in LLVM IR?
    - Aaron: closely
    - Joshua: LLVM IR can do more than what Clang exposes. Functions can be in different address spaces, for example; that's a definite problem for GPUs
    - Joshua: I'll be there next week. I have two pages of notes on the paper
    - Ariel: ptr32 is another problematic case
    - Aaron: yes, also ptr64, sptr32 and sptr64
    - Aaron: I just implemented the support to ignore those qualifiers, then someone else implemented them: https://github.com/llvm/llvm-project/blob/7b72b5fde414fd7d41653a07ddb87f80038009aa/clang/include/clang/Basic/Attr.td#L4489
    - Joshua: I'm not sure they are implemented as address space qualifiers in LLMV IR
    - Aaron: between me and Joshua I think we have this covered
    - Aaron: another paper is about adoption of qualifier conversion rules. Not sure if it achieves the goal, but not convinced it doesn't, too
    - Corentin: why do they want to have C++ rules with less rigor

    - Aaron: the paper is specified for all qualifiers. I'm worried what is going to happen to non-standard qualifiers
    - Ariel: why can't volatile be added?
    - Aaron: because accessing non-const via const is fine, but the same is not true for volatile
    - Aaron: C++ wording decomposes type into qualifiers, C wording is talking about adding and removing qualifiers
    - https://eel.is/c++draft/conv.qual
    - Vlad: I wish we had this meeting every week, so that we could discuss WG14 papers the entire hour. There is a paper that proposes to allow arrays of incomplete types — array itself would be an incomplete type, instead of ill-formed
    - Aaron: there is also contracts paper
    - Aaron: agenda is backwards, because we need to publish the next standard in 2029. Draft should be feature complete by the end of next year
    - Vlad: I did a second review of the embed sync paper and now it's good
    - Aaron: I think Jean Heyd moved it off the schedule due to late changes
    - Corentin: that's an important paper to get into C2y
    - Vlad: r2 had a lot of changes, r3 really only has one small change which is unfortunate
    - Aaron: it wll get discussed, through NB comments if needed.
- Do we want to deploy `-verify-directives` (PR) outside of C++ DR tests?
    - Vlad: Already have -verify for checking diagnostics; widely used in our tests
    - Vlad: over the years, I developed practices for how to write the expected directives for DR tests based on what works well for readability and tooling
    - Vlad: adding an opt-in mode to validate against those practices, -verify-directives
    - Vlad: it catches issues with things like diagnostics which are emitted out of order but still pass the usual validation because the correct diagnostic is emitted
    - Vlad: wondering if there's a reason to use this outside of DR tests
    - Aaron: (takes over presentation) it makes sense for DR tests, but I'm on a fence if we need this in other tests. For instance, sometime we omit diagnostic text when it's not relevant. I'd be fine with an encouragement, but less fine with a hard requirement
    - Corentin: asking for a regex every time we're not interested in diag text is
    - Vlad: currently documenting the mode and requiring it in C++ DR tests, the question is whether we want to require/encourage people to use it outside of DR tests. e.g., reviewer can ask the author to use this mode to make it easier to review the test, particularly in template-heavy code
    - Matheus: I would like this to mature before we encourage it to be used more
    - Vlad: current deployment is being used in DR tests, do you need something more beyond that?
    - Matheus: I think there's more to explore in terms of verifier improvements, so it's less of a moving target for changing tests

    - Corentin: idea, when we find a test that has a bug that's caught by this flag, we add the flag at that point to that test. I don't see this causing a lot of issues in practice during reviews, I think the status quo is fine for many years.
    - Aaron: would it be reasonable for author to say no if reviewer is asking for this mode?
    - Corentin: it should not be asked for big files. Create a new file for this new test
    - James: we use -verify outside of clang tests, for generic libraries. Overall it works super well, but some of the issues brought up there are seen. This solution might not be what we need. Errors might be emitted somewhere else from where template instantiations
    - Aaron: can I tag you on the PR? I didn't know anyone is using -verify outside of clang
    - Corentin: libc++ is using it
    - Corentin: one of the issues is that diagnostics are not related to each other
    - James: there is a flag to collect all diagnostics. If there is a error in a template, we don't know if template or its user is to blame
    - Vlad: I'm not sure if other improvements are in scope, but implementation was intentionally kept simple
    - Aaron: I'm hearing that folks don't mind use in C++ DR tests, but cautious about deployment beyond that
- Increasing number of "pure-AI" contributions from new contributors. Can we somehow stop wasting time on these? Maybe not allow new contributors use LLMs similar to not allowing using LLMs for good first issues
    - No assisted by label on these
    - Content is often not relevant
    - PR description is too long and does have Markdown that no human uses for commit messages
        - Sometimes they quote obsolete standards in a very unhelpful way (example)
    - https://discourse.llvm.org/t/concerns-about-low-quality-prs-being-merged-into-main/89748
    - Example https://github.com/llvm/llvm-project/pull/181579
- P4032R0 "Strong ordering for meta::info"
    - Corentin: this is mine
    - Corentin: I'm talking to more people, we have concerns about the draft. Meta::info is a structural type, so we can mangle then. I assumed we'll mangle the address of AST node. Meta::info can represent things that you can't have in a template parameter.
    - Aaron: there was a discussion in CAT, we recognize that we need to restart itanium abi group, with the understanding that it's us. We need to get some GCC folks on board
    - Corentin: I agree, but we need to understand what do we do with mangling in C++26. We had a long discussion about type ordering, and decided to use

mangled name. This paper proposes that order of types has to match order of their reflections ,which changes what we've spent a lot of time discussing.

- Aaron: you said that we're going to mangle reflection as an address, but it's not useful, because we can't compare for equality
- Shafik: can we check what GCC is doing?
- Corentin: one of the reasons I thought we can do that is because you can't observe meta::info outside of compile time. But you can have meta::info as a template argument, and this won't work across TUs
- Vlad: feature needs meta::info to be a template parameter to avoid ODR pits
- Matheus: I'm not sure what the big problem is
- Aaron: we can;t just look at GCC code because of GPL concerns
- Shafik: why do we need to mangle it if we can't use it outside of templates?
- Corentin: say we have an array of meta::info, and you want to sort it. Your sorting function has to be one function across Tus
- Aaron: whatever we come up with, when we implement it, we'll have to deal with whatever Microsoft is doing
- Proposed manglings: https://github.com/itanium-cxx-abi/cxx-abi/issues/208
- James: GCC implements this, but it is not stable yet

Meeting Agenda - February 18th, 2026

Agenda:

- Clang 22 is approaching RC4
  - 24th is deadline for RC4
  - Shafik: there are regressions linked to Matheus' PRs about resugaring
- Results are out for Clang Area Team 2026 elections
  - Less voters this time (thread)
  - First meeting this afternoon, in a few hours
  - Aaron, Eli, Corentin, Erich, and Shafik are members
  - Tags like `clang-area-team` point to the old CAT at the moment
  - Aaron: there are people who did not expect elections to be so soon, then there were wording issues with the titles of Discourse threads
  - Ambrose: I found out about it on the last day
  - Erich: doing over?
  - Aaron: no, we've got enough votes
  - Vlad: how this group will interact with CAT?
  - Aaron: I don't think anything will change. CAT is not as interested in language support. Extensions are an overlap, though
- WG14 meeting
  - Aaron: took place two weeks ago, when we were supposed to have the last meeting
  - Aaron: only 4-5 things voted in that are not issue resolutions
  - Aaron: nothing particularly contentious in the issue resolutions
  - Ambrose: is optional back?
  - Aaron: in a TS
  - Aaron: I'll probably write a paper that optional is not shippable even as a TS. sdcc decided to implement it, because they are the only who can get away with it. There are features that say that this pointer is not null. Then WG14 want to have a feature that pointer must not be null. Both markings are needed. Unless it can be incrementally adoptable, feature is not viable. This feature came up in our community first. We rejected, then the author went to WG14
  - Corentin: C thinks that everything has to be qualifier. _Atomic is bad enough, and I don't want more of that
  - Aaron: specifiers vs qualifiers doesn't bother me too much
  - Corentin: attribute would not pollute the type system
  - Erich: that attribute should be spelled "&"
  - Joshua: several of the committee members have issues with qualifiers and attributes, because they always want them to be droppable
  - Aaron: thankfully, optional was not discussed, but forward declared params were but did not get consensus
  - Vlad: not dead yet, he scheduled the paper and did not schedule the new revision of Yeoul's paper. So maybe coming up again.

- Aaron: Robert did not realize that Chris' paper had no new information, but made it the deadline for February
- Aaron: next meeting is scheduled on March
- Aaron: committee doesn't seem to be interested in a bunch of new helper functions for standard library. Committee's appetite for new features seems to go down
- Corentin: I"m not sure those helper functions help anything
- Aaron: old WG14 members chatted on mastodon, expressing their dissatisfaction with WG14 this cycle
- Aaron: I'm having 1-on-1 conversations to mend bridges between implementers and non-implementers. Hope is that couple of non-implementers will understand implementation concerns, reducing the contention in the committee
- Aaron: I'm having monthly meetings with Martin Uecker, who occasionally contributes to GCC, but otherwise works in biomedicine domain. Hope he'll socialize some of our problems. At some point if those 1-on-1 go well, I'll invite those folks to this meeting for extra perspective.
- Vlad: lamda discussion was a disaster
- Aaron: JeanHeyd came with a paper describing the design space, but committee was like "tl;dr". Immediately after we discussed other papers which went for half-solutions, but no decisions were made
- Joshua: JeanHeyd's paper was interrupted at the end, but had all the information.
- Aaron: I think there's misunderstanding between you and Martin, because he moved on from trampolines
- Joshua: even if you don't propose trampolines, but the way feature works, trampolines will be needed
- Corentin: is there an implementation of nested functions without trampolines?
- Aaron: opt-in, because of ABI
- Corentin: I talked to people at apple, and they are very concerned about security
- Corentin: users will be very confused if lambdas will come without captures
- Aaron: regex for lambdas is hard, but it seems that there are 3 times more lambdas with captures than lambdas without them.
- Aaron: C doesn't really need anything in this space. Rushing the solution is the real problem here
- Vlad: for context, some folks in WG14 think lambdas are perfectly viable without captures
- Aaron: any more questions about WG14?
- Ambrose: any updates on defer TS?
- Aaron: it wasn't on the agenda, but I'll make sure it is on the next one
- Aaron: JeanHeyd gave WG14 update on implementation experience. There was no reactions from WG14
- Corentin: have we got any feedback from our users?
- Aaron: no, because 22 is not out yet. I don't expect any bug reports, because it's just a syntax for attribute cleanup

    - Aaron: I have not heard about GCC implementation of defer. I know there's a patch under review, but it might be too late for stage 4
    - Corentin: reflection was pushed at stage 4 of GCC 16 release
- Remove 80 column limit in documentation files (RFC)
    - Joel created a script to review PRs via git diff (post)
    - Aaron: there is a poll. Please vote
    - Aaron: 80 column limit doesn't help in documentation
    - Vlad: there is an illustration in the thread
    - Ariel: back in the day there was a latex guideline to start every sentence with a new line
    - Aaron: this is where this option on the poll came from
    - Vlad: Mehdi is asking for tooling to enforce one sentence per line
    - Aaron: it should be trivial to implement
    - Corentin: some sentences might be very long
    - Vlad: we're not good at upholding the limit
    - Shafik: I changed my opinion after your office hours to one that is close to yours
    - Erich: will this be discussed in CAT meeting today?
    - Aaron: yes, but it's too early to call consensus on that
    - Hubert: I might be voting for a less popular options
    - Aaron: that's the beauty of it: everyone will be dissatisfied
- Hubert: custom clang-format configs per subproject
    - Hubert: I think they should die, because some of them violate our coding standards with regards to includes
    - Aaron: theoretically, every subprojects has to follow the same coding standards. Practically, this never was the case. MLIR picked their own coding standard, which causes friction now that ClangIR is integrated into Clang
    - Aaron: if our clang-format violates something, we should change it
    - Ambrose: can be disable clang-format for tablegen files?
    - Ambrose: adding //clang-format off at the top of every tablegen file might not be a bad idea
    - Aaron: for the long time we resisted tool-specific annotations
    - Ambrose: I agree on principle, but clang-format is the only tool we have this issue with
    - Corentin: we can disable it per directory https://clang.llvm.org/docs/ClangFormat.html#clang-format-ignore
    - Aaron: this has been annoying for the long time, would be nice to fix that
    - Ambrose: it's annoying for reviewers, too
    - Aaron: I feel terrible to tell people to ignore clang-format
    - Aaron: Hubert, since you're looking at this, would you look at td files?
    - Hubert: I could, but it can't be the same PR
    - Aaron: **I'll take an action item**
- Do we want to deploy `-verify-directives` (PR) outside of C++ DR tests?
- User branches without a PR will soon be deleted (thread)

    - Originally, user-created branches were expected to be deleted, but that work did not happen
    - Aiden is now starting to tackle this; will prune branches on a daily basis based on whether there's a public PR tied to the branch. They will archive the branches when deleting them so data is not lost.
    - Ariel: what's the point of the one-day delay?
    - Vlad: to avoid timing issues between creating the branch before the PR is created; having the delay makes that less likely
- Long-term vision for improving build times (Meta-RFC)
    - Some desire to make clang do more build system work; perhaps can do better caching or better module dependencies, use vfs instead of expensive windows file system calls, avoid process creation overhead, etc.
    - Very large RFC, but author has some short term goals
    - Shafik: this came up during the LLVM social, there was skepticism about this being viable. The biggest beneficiary seems to be Windows but there's quite a bit of risk for different ways to break things. It may be too ambitious, but a more narrow focus may gain traction. Questions of "who will maintain this long term"?
    - Reid: Previously worked with the author, talked in other contexts before. I think the project would benefit from a lot of the short-term goals. Already have to patch LLVM to avoid conflicts with global state (within NVidia). I don't see Windows as being someone else's problem; dylibs on linux are another example. Process startup times can be slow everywhere. Removing extra process launch time from RUN launching a custom shell still saves like 10% of time. Repitching the RFC with a narrow set of goals could be successful.
    - Hubert: Reid got to most of what I was thinking; I think this could be helpful for more than just Windows.
    -
-
- Next agenda:
- Do we want to deploy `-verify-directives` (PR) outside of C++ DR tests?

Meeting Agenda - February 4th, 2026

Agenda

- Historical notes removed from this document and archived on Discourse
    - https://discourse.llvm.org/t/historical-clang-language-wg-meeting-minutes-mar-2025-jun-2025/89546
    - https://discourse.llvm.org/t/historical-clang-language-wg-meeting-minutes-jun-2025-sep-2025/89547
    - https://discourse.llvm.org/t/historical-clang-language-wg-meeting-minutes-sep-2025-dec-2025/89548
    - Required three separate posts due to character count limits. We talk a lot. ;-)

Meeting Agenda - January 21st, 2026

Agenda

- rc1 is out, rc2 is expected next Tuesday (27th). Final release is expected Feb 24
    - Aaron: if there are regressions, they should be prioritized, especially regressions introduced in 22
    - Aaron: release coincides with two WG14 meetings: one in february, one in march right after the final
- WG14 (virtual) meetings happening Feb 2-6 and Mar 9-13
    - Preliminary agenda: https://www.open-std.org/jtc1/sc22/wg14/www/docs/n3790.pdf
    - Quite a few pure inventions (N3694, N3678, N3679, N3674, N3454, N3692, N3458), if you see papers you want me to express feedback on, please let me know (on any paper, not just these)
    - Aaron: my availability is going to be random. Has to read papers and be present in the meetings. But meetings are just half a day.
    - Aaron: preliminary agenda for WG14 meeting is out.
    - Aaron: if you have feedback on papers, let me know
    - Aaron: pure inventions make the most sense to review
    - Aaron: there is a big block of papers about lambdas-like facilities
    - Ariel: what do you mean by fat function pointers
    - Aaron: the way to add more information to functions
    - Corentin: would be nice to have a pointer type
    - Aaron: there are bugfixes for C23 features, like removing UB when no one is exploiting it
    - Hubert: papers are poorly names. UB is a symptom how C writes wording
    - Aaron: UB SG is getting rid of any stupid UB, leaving only meaningful UB. paper titles are indeed are not super helpful
    - Ariel: are fat pointer going to have a new linkage?
    - Ariel: linkage describes how functions are called
    - Hubert: they are more like function descriptors, holding additional information to establish the context, more like a closure
    - Aaron: that's all for WG14 meetings. If you have questions, reach out to me
    - Aaron: if I see something that desperately needs attention, we have a clang wg meeting right in the middle of WG14 meeting
    - Corentin: if any UB papers touch lexing or preprocessor, let me know. In the past C made changes that are incompatible with C++
    - Aaron: looking through the list. So far not seeing anything that matters. THere is a paper that refactors grammar for preprocessor directives
    - Vlad: Alejandro brought this paper in Core. JEns was not impressed with the formatting
    - Hubert: just for the formatting. We're going to apply it to C++, too
    - Aaron: I see your papers on the list
    - Corentin: I'll be there

    - Hubert: my issue with that paper is how POSIX people are going to deal with it, like on fork().
    - Aaron: you guys can talk about this offline. THere are folks from POSIX in the meetings, so they should be represented
    - Ariel: if fat pointer is a funticon descriptor, why this has to be a language feature?
    - Aaron: trying to do what lambdas do in C++. Closures are needed. Two competing approaches are C++ lambdas and GCC nested functions. There is a a third approach
    - Hubert: block?
    - Aaron: surprisingly, they have not been proposed
    - Corentin: we still have implementation objections to GCC implementation of nested functions
    - Aaron: yes, trampolines and executable stack
    - Hubert: since you've came up with this list. There is a paper about language linkage blocks. Why is it problematic?
    - Aaron: there have been disagreements about what those blocks mean
    - Aaron: if I have extern block, I should be able to use static on array extents — kind of confusions people have
    - Aaron: so this is inventive. A lot of inventive stuff come from people without implementations, out of pure enthusiasm, but it costs us money.
- Implementation reality of WG21 standardization P3962R0
    - Hubert: there is WG21 paper about implementer feedback
    - Hubert: there is new version of P1000, which is IS schedule. Wonder how those papers will be discussed
    - Aaron: at the Kone WG21 meeting, there was an evening sessions for people who self-identify as implementers, both for compilers and library. The output of P3962R0, which is a good distillation of the discussion. There is a lot of concern from all implementers how expensive it is to implement the language. A lot of experiments which we need to pay for. We need deployment experience, not just implementation in a fork. Paper is a list of requests to the committee. Like CWG and EWG running simultaneously means that implementers has to choose where to be. I don't know where the paper is going and what the impact is going to be, but I'm happy to see we're not alone thinking that LLVM is drowning in work
    - Shafik: there was also discussion about last C++ version an implementation is going to support. Louis had an idea for a paper to make C++29 a bugfix release.
    - Aaron: I've taking a liberty of alerting WG14 convenor of this paper. We have the similar kind of concern for WG14. Not sure what they will going to do, but would be nice to bring this up in front of the WG14, even if C features are not as expansive as in C++. There is a proposal to bring contracts, but C++ committee itself doesn't understand the feature
    - Hubert: there are people who understand, but (even within that subset) they do not agree whether trade-offs are right
    - Aaron: Jens is aware of C. contracts with come up in headers, so it's a compatibility concern

    - Corentin: I'm sad I didn't put my name on the C++ paper. Was busy with other stuff
    - Hubert:
    - Shafik: it was a therapy session
    - Corentin: impelementations do not agree on everything, but this is fine
    - Aaron: speaking of relfection, GCC put reflection in stage 4, even though stage 3 was not supposed to get new features. Wonder how it goes for them
    - Aaron: this gives us something to test against, at least
    - Michael: they are getting bug reports for it (https://gcc.gnu.org/bugzilla/buglist.cgi?quicksearch=reflection)
    - Aaron: I don't have anything else on P3962R0
    - Hubert: Nina was focusing on things we have solutions for, not everything we discussed.
    - Vlad: Nina also did not put stuff that had solutions, like tests with features
    - Hubert: yeah, but we should not dilute the existing points
    - Vlad: that's why I argued that the message should be that this is just the beginning of our complaints
    - Shafik: I also tried to get it into the paper (both that proposals need to have more rigorous tests and that we need a test suite)
    - Aaron: committee doesn't want to compete with its members, but we need the tests
    - Shafik: this bites us over and over again (that we miss poor interactions or breaking changes b/c we did not have sufficient test cases)
    - Aaron: if WG21 say "we can't", but we still need this, then we need to collaborate with other implementers on a shared test suite.
    - Shafik: there might be licensing issues
    - Aaron: we could've put new tests open source
    - Aaron: we've spent 300 hours on reflection, but didn't get tests out of it
    - Vlad: I've asked Dan several times to put test cases, and it's under our license
- Maintainer List Refresh
    - Verified 32 maintainers in Clang, still waiting to hear from 13
    - If anyone is interested in stepping up as a maintainer, please talk to me about it, it's easy and fun and you'll never regret volunteering to do more work!
    - Aaron: I reached to people privately, which turned to be blessing and a curse. Some had wrong emails, which is now fixed. But 30% did not respond.
    - Aaron: most still continue being maintainers, some step down. We're having additional categories, like coverage. Corentin volunteered to maintain concepts. If you want to be a maintainer of a new area, let me know. If you know someone is doing this work without recognition, nominate them after speaking to them about it. I'm also doing refresh in clang-tools-extra, but it goes slowly. Searching for lead maintainers for projects that do not have such. Found maintainers for libclc,libsycl, libcxxabi, openmp. No lead maintainer for lld, mlir. THere are active maintainers, but sometimes projects do not lend themselves to a position of lead maintainer. I'll work with project council to figure this out

    - Matheus: elf and coff linker used to be entirely separate path, down to copying parts of implementation
    - Aaron: yes, they are three linkers there
    - Aaron: there are times when we want more than one contact point
    - Matheus: my employer is sponsoring students to work on clang, like GSoC.
    - Matheus: for example SARIF. Is someone working on it?
    - Aaron: I did the initial work to bring it for clang sa
    - Aaron: production quality? Maybe, maybe not
    - Matheus: for the compiler, not just static analyzer
    - Aaron: sarif output is getting used, static analyzers integrated clang's sarif output, like msvc
    - Aaron: at least you can get diagnostic fidelity out of it. SARIF lets you have code path inside diagnostic, insights into analysis-based diagnostics
    - Aaron: it's easy to make improvements, because it's already there
    - Ambrose: https://github.com/llvm/llvm-project/pull/174106
    - Matheus: the main thing is to put proposals like GSoC
    - Aaron: if SARIF is important for your employer, we can set up a maintainer for it. Let's pretend you are maintainer. It's easier for you to work with lots of students when you have experience
    - Matheus: #embed is also interesting. Anyone working on it?
    - Aaron: Maria is working on it.
    - Aaron: if you're looking for topics, improve compile times
    - Matheus: I'm working on this
    - Matheus: we're looking more into GSoC level of things
    - Michael: there's a bunch of people in bloomberg who want to look into this. Matheus, we should chat about collaborating
    - Aaron: I can help coordinating
    - Aaron: we have 5 minutes left
    - Hubert: refresh minutes and archive minutes from 2025
    - Hubert: there is a small PR (https://github.com/llvm/llvm-project/pull/176551). Are we going in the right direction? WG14 has a thread about this. People sometime need this without the full breadth of MS extensions. Cc1-only option is a way
    - Aaron: I was planning to review it this week
    - Aaron: this should be driver flag, like "is char signed"
    - Aaron: I can see this useful enough that users would want to control this separately
    - Vlad: we shouldn't do this as cc1-flag
    - Aaron: my understanding is that it's for downstream, not directly exposed to users
    - Hubert: there are going to be users who won't complain about -Xclang
    - Aaron: I'll post on Discourse if something changes for the next meeting.
- AI policy has landed.

- Aaron is excited. Good incremental progress, but is not going to move the needle for quality of submissions
  Shafik: agreed
- Hubert: hope it moves the needle
- Aaron: it's awkward to assert that something is extractive
- Ariel: what is extractive?
- Aaron: asking community to do more than you did
- Corentin: not related to clang, but there is an RFC about using AI for Bazel, because it goes into the opposite direction.
- Aaron: yes, I hope we don't make exceptions. Peripheral tier should not get exceptions. But I don't think Bazel should exist, so I might be biased. There is a burden on the community to maintain it, which RFC now tries to address
- Hubert: (muses that the Bazel proposal is drawing lots of energy because some people are afraid of a "slippery slope", but personal not sure it is that dangerous)
- Aaron: we'll iterate on the policy as we learn. It's still good incremental progress
- Hubert: I followed the AI policy thread. I was not aware that ms178, the person who did AI stuff in mesa, also did AI stuff in LLVM

Meeting Agenda - January 7th, 2026

Agenda

- Vlad: reflection meetings start on Jan 9th on a two week cadence (post)
- Aaron: Clang 22 branch date is Jan 13th (next Tuesday)
    - There are discussions about switching to April/October releases but not for 22
    - Speak to release managers if interested
    - Current thread that started the discussion: https://discourse.llvm.org/t/rfc-freeze-the-llvm-library-abi-with-rc1-starting-with-llvm-23/89373
- Vlad: “Implement P1857R3 Modules Dependency Discovery” (PR)
    - Relands 1, 2
    - Taken a couple of tries to get this landed, still not quite ready. Impacted downstreams quite a bit.
    - Probably not making it to Clang 22
    - There was a maintainer suggesting that the author land the changes due to the PR being open for a while.
    - Still would have been better to hold off until there were more eyeballs
- Aaron: “Not assuming there is at most one definition in a redeclaration chain” (RFC)
    - Aaron: hope we’re not landing this in 22, left some comments
    - Aaron: seen overflow behavior discussions, but they are not targeting 22
    - Aaron: going to prioritize recent regressions
    - Aaron: if you see issues after branches, please prioritize those recent regressions
    - Aaron: this RFC is a heads-up for both upstream and downstreams
    - Aaron: for the longest time we’ve been assuming there’s only one definition in the declaration chain, but this is not true in both C and C++
    - Aaron: we’ve been fixing this here and there while working on modules
    - Aaron: can have impact on downstreams and plugins
    - Ariel: multiple definitions?
    - Aaron: yes
    - Hubert: is this only AST level?
    - Aaron: I think so, should not have backend impact
    - Vlad: C++ is modules, why is this problem in C?
      Aaron: in C this is only for a type. We have a rule type compatibility rule between TUs, now we have it intra-TU for type definitions.
    - Hubert: C scoping rules for types are different from C++.
    - Aaron: in C you can define a type everywhere you can name a type, which is not case in C++, e.g. in a compound literal. Change in C made it less surprising for macro users
- Hubert: is there a thread on Discourse about release dates that I can follow?
- Aaron: yes: thread
- Shafik: I’d like to complain about a spree of fuzzing issues
- Shafik: there is a particular user who filed low-quality issues

- Shafik: npopov handled him
- Shafik: when I see a new user filing fuzzing issues, I explain to them how to do that in a useful manner. It takes time, I'd like to see an official policy
- Aaron: <talking while on mute>
- Aaron: it'd be good to have a policy
- Aaron: in order to keep the policy welcoming, ask people to talk to us on discord before starting
- Shafik: that would be nice
- Shafik: my manager is asking whether this is university work, companies or just individuals
- Aaron: then we can guide them before everyone is frustrated. But of course not everybody read policies
- Shafik: at least we don't need to explain the same thing over and over again
- Aaron: you've got a hold of that. I was keeping an eye on your interactions on the bug tracker
- Round table:
    - Aaron
        - Getting back, digging out
        - Catching up on code reviews and mentions
        - If you need my attention for priority, poke me manually
        - Administrative stuff: reflection meeting, project council meeting, CoC meeting
    - Ariel
        - Hexfloat, hoping to upstream shortly
        - Aaron: that's not the parsing part, right?
        - Ariel: different representation
        - Ariel: there's going to be a PR to support hexfloat in APFloat, then another PR to support hexfloat in frontend
        - Aaron: I see
    - Hubert
        - IBM is actively working -finput-charset and -fexec-charset
        - For -finput-charset, I'd like to confirm something with the group.
        - z/OS requires to have pragmas to specify code pages
        - Encountered a lot of troubles with source location and diagnostic infrastructure
        - It feels that adding a layer of indirection to that infrastructure is going to kill performance. Is that correct?
        - Aaron: yes
        - Hubert: we need to reconcile z/OS and non-z/OS approaches
        - Aaron: terrible idea: pre-preprocessing for those code page pragmas
        - Hubert: you could mean two things. First approach needs to be a real pre-preprocessor
        - Aaron: yes, that is what I had in mind

- Hubert: we have people internally who believe this is the way. Otherwise, if we do this in the actual preprocessor, sourcemanager already has a concept of source locations that is not compatible
- Aaron: you could support pragma downstream
- Hubert: interesting idea. Not sure what z/OS stakeholders think of this. If we could isolate it properly, it might be fine to upstream. Depends on the shape of things. Don't want to start with a downstream-first approach
- Ariel: I think there's the case to upstream this. Users want to adapt upstream to z/OS
- Aaron: my gut feeling is that pragma is so weird that upstream will not be comfortable. Needs explaining how prevalent the use case is. Rajan explained to me that this can appear in system headers
- Hubert: yes, if you use national characters in your source, you'd have to use the pragma
-
- Hubert: -fexec-charset has issues with formatted output
- Hubert: one has to do with the source locations and diagnostics. Can't easily map source location to the souce file when -fexec-charset is used
- Hubert: there's divergence between implementations on conversion specifiers when multi-byte encodings are involved. When you hit percent sign, you drop down to code points, then go back to multi-byte. C's concept for string might not be adequate and what users expect. Linux seems to work fine.
- Hubert: practical effect is that I don't think we want multiple string interpretations.
- Aaron: we can follow GCC behavior and try to defend it
- Hubert: GCC has bugs they haven't fixed in decades
- Aaron: when did the C committee doubled down on the current approach?
- Hubert: before C23
- Aaron: because there's divergence on the platform level, it's not necessarily on implementation level.
- Hubert: I don't think disagreement was intentional
- Aaron: is this an issue at the OS level?
- Hubert: yes. There's a limited number of C libraries that support locales but don't deal in UTF-8
- Aaron: we might need a paper saying that the resolution to that DR is not implementable
- Hubert: but C libraries implement the C standard
- Hubert: OS folks need to come to the WG14 and defend what they currently do. We need to double down on UB
- Aaron: committee is not going to like that. Llvm-libc folks are starting to attend WG14 meeting, but they are not going to support locales
- Hubert: UTF-8 makes it hard to negotiate

        - Hubert: the clarification that WG14 made is that character means byte, which doesn't work with UTF-8
        - Aaron: let me know if I can help with WG14
        - Hubert: when we know what we're going to do, I'll reach out to you, because we have more than one issue
        - Aaron: not sure what the community thinks. Too esoteric
    - Islam
    - Mariya
        - Got back after holidays
        - Digging out of emails
        - Want to get back to fixing static variables in consteval functions
    - Nikhil
    - Shafik
        - Fuzzing
        - Soft-off: screening issues
        - Dealing with static analysis internally
        - We should be following the rule of three, because RAII wrappers don't need copy ctors and copy assignment operators
        - Does this need an RFC?
        - Aaron: RFC makes sense, because there are rule of zero and rule of five, but hopefully not contentious
    - Sharath
        - Don't have anything to report
        - Any guides to get started?
        - Aaron: welcome! If you're on discord, go to #clang and look at the pinned message
        - Sharath: I'll go through it and ping you
        - Aaron: works for me
    - Tony
        - Continue to work on fixing attributes using auto
        - Triaged and found around 6 attributes that need to be fixed
        - As reported in the issues about cleanup attribute, I fixed something for objc
        - https://github.com/llvm/llvm-project/pull/164440
    - Vlad
        - Mostly off
        - Reviewed couple of PRs related to python bindings
        - Don't feel comfortable reviewing things related to packaging and deployment, could use someone with expertise to help
        - Aaron: I'll see if I can find someone with expertise to help