Draft Minutes for 16 May – 20 May, 2022

MEETING OF ISO/IEC JTC 1/SC 22/WG 14
WG 14 / N 3004

Dates and Times:

Monday,	16	May,	2022	13:30 – 17:00 UTC
Tuesday,	17	May,	2022	13:30 – 17:00 UTC
Wednesday,	18	May,	2022	13:30 – 17:00 UTC
Thursday,	19	May,	2022	13:30 – 17:00 UTC
Friday,	20	May,	2022	13:30 – 17:00 UTC

Meeting Location

Teleconference

1. Opening Activities

1.1 Opening Comments (Keaton)

1.2 Introduction of Participants/Roll Call

Name	Organization	NB	Notes
Aaron Ballman	Intel	USA	C++ Compatibility SG Chair
Alex Gilding	Perforce / Programming Research Ltd.	USA	scribe
Barry Hedquist	Perennial	USA	PL22.11 IR
Clive Pygott	LDRA Inc.	USA	WG23 liaison
David Keaton	Keaton Consulting	USA	Convener
David Svoboda	SEI/CERT/CMU	USA	Undefined Behavior SG Chair
David Vitek	Grammatech	USA	
Elizabeth Andrews	Intel	USA	
Fred Tydeman	Tydeman Consulting	USA	PL22.11 Vice Chair
Freek Wiedijk	Plum Hall	USA	
Lars Bjonnes	Cisco Systems	USA	
Maged Michael	Facebook	USA	
Martin Sebor	IBM	USA	
Nick Dunn	NCC Group	USA	
Rajan Bhakta	IBM	USA, Canada	PL22.11 Chair
Robert Seacord	NCC Group	USA	
Tom Honermann	Intel	USA	
Aaron Bachmann	Austrian Standards	Austria	Austria NB
Bill Ash	SC 22		SC22 manager
Dave Banham	BlackBerry QNX	UK	MISRA Liaison
Eskil Steenberg	Quel Solaar	Sweden	Sweden NB
Hubert Tong	IBM	Canada	
JeanHeyd Meneide	NEN	Netherlands	Netherlands NB
Jens Gustedt	INRIA	France	France NB
Joseph Myers	CodeSourcery / Siemens	UK	UK NB
Marcus Johnson		USA	guest
Martin Uecker	University of Goettingen	Germany	
Michael Wong	Codeplay	Canada, UK	WG21 Liaison
Miguel Ojeda	UNE	Spain	Spain NB
Niall Douglas	Ireland ned Productions Ltd	Ireland	guest
Reini Urban		Germany	guest
Thomas Koppe		USA	guest
Peter Sewell	University of Cambridge	UK	Memory Model SG Chair
Philipp Krause	Albert-Ludwigs-Universitat Freiburg	Germany	Germany NB
Roberto Bagnara	BUGSENG	Italy	Italy NB, MISRA Liaison
Ville Voutilainen	The Qt Company	Finland	Finland NB

1.3 Procedures for this Meeting (Keaton)

We hold straw polls instead of formal votes; guests may vote.

We do *not* want multiple conversation threads; do not split the audio and chat. Do not have more than one conversation at once. The chat is an additional medium for backup and enhancements (such as URLs) only. The chat is not minuted unless being used as a substitute for audio.

1.4 Required Reading

1.4.1 ISO Code of Conduct

1.4.2 IEC Code of Conduct

1.4.3 JTC 1 Summary of Key Points [N 2613]

1.4.4 INCITS Code of Conduct

1.5 Approval of Previous WG 14 Minutes [N 2803] (WG 14 motion)

Tydeman: found some errors. Fixed by Bhakta, no major changes except that the wrong dates were on the paper deadlines. No objections to amendment.

Ballman moves, Svoboda seconds, no objections.

1.6 Review of Action Items and Resolutions

DONE: Meneide: Write a policy paper for naming of newly proposed identifiers for the standard.

DONE: Keaton: Look into what paper to add to the papers-of-interest list.

DONE: Keaton: Add C charter discussion for the next meeting.

We said to add N2761 to the list, but that was a mistake. What we intended was to put the final version of that paper on the list if it's accepted. The author of the paper is not a WG14 member, so Keaton asked the author to take an action item to do that work when they have the final revision of the paper.

ACTION: Marcus Johnson to to add a request to place the final version of the Unicode Length Modifiers proposal on the Papers of Interest list.

There was a second paper that should be on the papers of interest list on assertions from Peter Sommerlad. Keaton talked to the author and the author was in favour of adding the paper to the list. The paper is N2829, the user-friendly assert macro, adopted into C23 at the previous meeting.

ACTION: Aaron Ballman to put N2829 on the Papers of Interest list.

1.7 Approval of Agenda [N 2990] (PL22.11 motion, WG 14 motion)

Tydeman observed a minor issue with an incorrect date in Section 9.

Combined motion: Seacord moves, Pygott seconds.

Bhakta would like an earlier agenda for the next meeting; a week was insufficient. Keaton agreed to do that next time. Myers was wondering what the priority order is for other business papers (if we

have time for them). Keaton said first priority is homework. Second priority is 7.1. Third priority is the rest of the list, in order, except for the people not attending the meeting. 5.2 on the agenda is being moved from Monday due to a scheduling conflict; we discuss 7.1.3 on a TS proposal in its stead.

Unanimous consent to adopt as amended.

1.8 Identify National Bodies Sending Experts

Austria, Canada, Finland, France, Germany, Italy, Netherlands, Spain, United Kingdom, United States.

1.9 INCITS Antitrust Guidelines and Patent Policy

Guidelines were observed without discussion.

1.10 INCITS official designated member/alternate information

Check with Bhakta if unsure.

1.11 Note where we are in the C23 schedule [N 2864]

Deadline for the final version of all proposals is Jun 17; Last proposals to change C23 at all is July 22; CD ballot in November 2022; Ballot resolution will happen in Jan 2023. Updates and editorial review to happen after.

Gustedt: last time we had an “intensive review”; where does that fit in the schedule?

Keaton: that will happen during Editorial Review (Aug 15) before the CD ballot, and then another one before the DIS ballot (Feb 13 2023).

Gustedt: would feel more comfortable if we had seen the versions long before that, because there's only a week to do this. Can the editor get an additional version of the document out around August 2022 so we can pass it out as homework for review?

Meneide: that's entirely doable, commits to doing that.

Keaton: the schedule may need to be lengthened a little bit, but even if we don't, we'll still need to ask for an extension at the next SC22 plenary. The extra influx of proposals at the deadline caused us to slip the schedule. We should still be able to call it C23 as we plan to be finished in Nov 2023. We only get one extension, so we have to use it wisely.

Would the committee like to request an extension at the next SC22 plenary?

Seacord: how long is the extension?

Keaton: it's a nine month fixed extension. The extension would get us to Jul 2024, but we don't want to use all of it.

Seacord: asked for the extension. No objections.

ACTION: David Keaton to request an extension to the C23 schedule at SG22 plenary.

2. Reports on Liaison Activities

2.1 ISO, IEC, JTC 1, SC 22

Keaton: ISO, IEC, and JTC 1 all changes the policy to no longer forbid hybrid meetings. We can have fully remote meetings or hybrid meetings, where remote attendees are on an equal footing with the in-person attendees. Fully in-person are still forbidden; you can get an exception if needed. Implication of that: hybrid requires everyone to be on an equal footing, so has to last half as long (same as virtual meetings) because that would cause problems for the remote attendees.

There's now a lot of scrutiny about guests invited to WG meetings coming from JTC1. We should not give out meeting credentials to anyone who is not a WG14 meeting, or officially invited by the convener.

2.2 PL22.11/WG 14

Bhakta: PL22.11 is now known as PL22.C (INCITS is doing that for other groups as well, like PL22.C++). Updated scope that we voted on was approved by INCITS, but they've not done the full change on the website (they did half the changes). Bhakta is coordinating with them to get it fixed.

2.3 PL22.16/WG 21

Ballman: WG21 meeting in Kona this winter for hybrid meeting; will be interesting given the requirements for equal footing.

2.4 PL22

Keaton: nothing to report.

2.5 WG 23

Pygott: WG23 is still working on the C++ language specific document and trying to get it published as a freely available IS or not. Keaton is continuing to work on that at the JTC1 level to get wider availability of freely available documents (aka, get them back to where they used to be).

2.6 MISRA C

Gilding: No formal liaison statement from MISRA, but informally we had an in-person meeting (with remote attendees) a few weeks ago. Amendment three is coming out soon (MISRA 2012 brought up to C11), announced but not released next month. Not certain if there will be another major release of MISRA or not.

2.7 Austin Group

No update.

2.8 Other Liaison Activities

None.

3. Study Groups

3.1 C Floating Point Study Group activity report

Continuing to handle questions and bugs from the community. Working to update 18661-4, the parts that were not added into C2x are being updated. Adding augmented arithmetic from 2020 IEC 60559, so part 4 will get new functions to raise conformance level.

There was an external request (to CFP and WG14) to make 7.12.13 and Annex F.10.10 to change from "floating multiply add" to "fused multiply add" to better match industry terminology. Bhakta is wondering whether this an editorial change, or does the requester need a paper and a champion?

Bhakta: in general what is the process when a change request comes directly to the CFP SG instead of to WG14? No prescribed course of action.

Meneide: fine making the change editorially, assuming everyone is okay with that.

ACTION: JeanHeyd Meneide to make editorial change from "floating multiply-add" to "fused multiply-add".

3.2 C Memory Object Model Study Group activity report

Hasn't had much activity to speak of, wants to move ahead to DIS ballot for TS 6010.

Sewell: wondering if changes to the TS contents are problematic for moving forward. We'll discuss further on Tue with item 5.5.

3.3 C and C++ Compatibility Study Group activity report

Met three times since the last WG14 meeting, continues to make progress on proposals for C23 and C++23. New time slot has issues getting quorum.

3.4 Undefined Behavior Study Group activity report

UB SG is also having troubles with quorum; time changes for DST also caused issues.

Holding off on submitting documents until C23 is finished. Hope to improve Standardese to avoid listing “compile errors” in Annex J.2.

4. Future Meetings

4.1 Future Meeting Schedule

Keaton: Next meeting is Jul 18-22 and is virtual. The meeting after that depends on the C23 schedule, but we'd like to time the meeting to be between the end of the CD ballot and the beginning of the DIS ballot. Aiming for late 2022 or early 2023, depending on the ballot schedule (which also depends on the extension). We'll set a date at the next meeting.

4.2 Future Mailing Deadlines

June 17 is the mailing deadline for the final proposals for C23 -- any revisions after that are only due to homework arising during the meeting.

Bhakta: clarifying, this is only for revisions of proposals already in the pipeline.

5. Document Review

Monday

5.1 Working draft updates [N 2912] [N 2913]

Meneide: has merged all of the outstanding papers and is currently cleaning up the output. The document will be published as N2912 once that's cleaned up. Also fixed up a lot of the build system dependencies, so it should be easier to build from source.

If you sent an editorial fix, almost all of those should be handled by now, so if you know of one that's missing, let him know.

Annex H “Language-independent arithmetic” is being replaced by Annex H from TS 18661.

5.2 Discuss how we wish to apply the C23 Charter [N 2611] (new additional paper by Sebor [N 2986])

Deferred to Friday (Sebor is not available).

In its place:

7.1.3 Gilding, TS proposal: C - Extensions to support pure functions [N 2976]

Gilding: the document is kind of a placeholder. This attempts to imitate the Embedded C TR, but as a TS in terms of form and layout, referencing N documents for the functionality. The content of the TS is `constexpr` (N2917), tail-call elimination (N2920) and the `constexpr` specifier for objects (N2954). This TS is helpful for shaking out implementation concerns where these features cause problematic interactions in the rest of the language. It's 90% about process rather than content right now, all this content can be changed.

Gustedt: a bit reluctant about the title; "pure" is a loaded term with attributes and means different things to different people. Perhaps we can have a more clear title.

Gilding: what we vote on the title right now actually matters, so we need to solve that before we can get the TS started.

Bhakta: like the idea of doing it this way; would have been nice if we had the charter discussion first because of the implementation experience questions. Fan of TS in general, want to see more done like this. Seeing someone implementing this TS successfully would make me vote to bring the functionality into a future version of C.

Gilding: "C - extensions to support generalized function calls" is the other title I was considering.

Gustedt: I like that title much better.

Seacord: Wondered if we actually have plans post-C23; that discussion was never concluded.

Keaton: Folks have started to call it C2y, but we'll have that discussion this week. To have a TS, we need five countries to commit to supply an individual to participate. That means you agree to read the drafts and comment if necessary. Which national bodies are willing to list a specific person to participate?

Austria (Aaron Bachmann) says yes.

Canada (Rajan Bhakta) says yes.

France (Jens Gustedt) says yes.

Germany (Martin Uecker) says yes.

Italy had no response.

Netherlands cannot commit at the moment.

Spain had no response.

UK (Joseph Myers) said yes.

US (Rajan Bhakta) said yes.

Keaton: any objection to put forth a NWIP for this TS? (none).

ACTION: David Keaton to submit a NWIP to SC22 for "C - Extensions to support generalized function calls".

5.3 Svoboda, Checked N-bit Integers [N 2867]

Svoboda: came from the mixture of `_BitInt` and checked integer safety proposals that came into C23 at the same time last summer, but we didn't think about the interactions between the proposals. They do interact after all as the existing text doesn’t exclude N-bit integers. Would like to see a clean interaction but could start by being more cautious.

The paper has some proposed wording to disallow bit-precise integers from the checked integer functions.

Gustedt: the wording you have makes it UB to pass a bit-precise integer, which means implementations can extend the behaviour. But we could also make it implementation-defined to force the implementation to document it.

Ballman: opposed to implementation-defined, that obligates everyone to do something. In favour of the paper as-is.

Seacord: ditto, keep it as UB to allow for implementation extensions

Straw poll: (opinion) Does WG14 want to adopt n2867 as-is into C23?

Result: 19-1-2 (adopted)

Uecker: why is it UB?

Gustedt: it's in the semantics section and uses a "shall".

5.4 Svoboda, Towards Supplemental Integer Safety [N 2868]

Svoboda: this is an updated paper; core proposal was accepted at the June 2021 meeting, this is the supplemental portion that was not accepted then. Addresses comments from the reflector, and also clarifies that checked integers do not support bit-precise integers, in advance this time. The wording changes replace the current wording in the standard. There's a checked integer type for every

unchecked integer type that isn't explicitly excluded. It doesn't specify the exact return types of the function (could be a struct, but doesn't have to be).

Gustedt: p2 says that these types are integer types; that doesn't seem to be what you mean (that means you'd be allowed to use operators on them, which doesn't seem like).

Svoboda: correct, you have to use the macros but not the operators. So probably the wrong wording that we can fix offline.

Krause: what about `intmax_t`? Don't we want to phase out interfaces with that type?

Svoboda: we weren't when I first started writing the proposal, so I thought this was fine. But we could remove them from the table.

Krause: even if we don't phase out old ones, I think we shouldn't add *new* interfaces that use `intmax_t`.

Ballman: agreed, no new interfaces with `intmax_t`.

Svoboda: fair enough.

Svoboda: if the implementation wants to add a new unchecked type, they're required to add a checked type. It adds constraints on the unchecked type.

Seacord: "checked value indicates..." happens in a number of places, but it doesn't mention things like wraparound. I have not quite nailed down what's being detected by the current wording. I think it's a bit closer to the result not being the same as the result not being reached in infinite signed precision arithmetic. Should we rephrase in those kinds of terms?

Svoboda: I've not seen anything about infinite precision arithmetic wording anywhere else in the standard. It's a bit of an academic question if you can have overflow, truncation, or misinterpretation of sign when you take infinite precision into account. I'm sympathetic to overhauling the phrasing, but that's a wider scope than this paper.

Seacord: "misinterpretation of sign" is also not wording we use in the standard, so there's already some invention of terms. I don't think what's here is correct; at least, I can't read it. It'd be helpful to create some new term of art and use it everywhere we spell this out long-hand. I'd want to see it corrected before adopting the proposal.

Svoboda: right or wrong is less important than whether it conforms to what we're doing in the standard.

Seacord: I don't have an issue with the rest of the standard, my concerns are specific to this new wording in this paper.

Gustedt: compromise would be to say like "the mathematical value does not fit within the range of representable values"?

Svoboda: modular arithmetic is also mathematical, so it's a problem.

Seacord: we could repeat exactly the language from the function description about what causes the overflow flag to be set.

Svoboda: we could use the same wording as before, so if that is correct, it's correct everywhere, and if it's incorrect, we can fix it all at once.

Seacord: that would address my concerns.

Svoboda: there are getter methods to get you the overflow information and the value information. There are no setter methods, so you cannot change the values or properties; only compute new values. There are some constructors for creating a checked integer type given a value and an overflow flag.

Gustedt: the function should not have a `_t` in 7.24.3.

Svoboda: that's a formatting error, it should be `ckd_mk_int`, not with the `_t`.

Svoboda: The paper also proposes `ckd_`*`op`* macros for `add`, `sub`, and `mul` (but not `div`). There's a three-argument form (where the result is a parameter) and it returns the overflow flag; and a two-argument form which returns a `ckd_type_t` type.

Seacord: the clarifying integer terms paper made some changes to the language that aren't reflected in the paper, so there may be a merge conflict.

Svoboda: yes, they were not captured here.

Seacord: looking at p7 on the top of page 17 has “wraparound” misspelled. But you'll go back and correct those uses?

Svoboda: I'll look at the core proposal to make sure it matches what we've got in the standard, so it'll be consistent.

Gustedt: I like the proposal as-is in terms of the semantics, but there's a lot of wording issues that need to be addressed. I sent many via email. p6 is new and needs to be rewritten. I can help you with that.

Bhakta: largely agrees with what's already been said; the wording doesn't seem to be ready yet.

Ballman: what's the implementation experience with the type generic macros? Are users happy with it?

Svoboda: the core proposal has implementation experience, but the supplemental proposal has no implementation experience.

Seacord: someone released an implementation of this on GitHub about a year ago, but no idea how well used the project is. I agree with Gustedt about liking the idea of the proposal, but the wording issues are a concern especially given that we're out of time with the C23 schedule. I like the idea of having a small editorial group that helps that the next time we see this will be the last time we see it so it can go into C23.

Svoboda: if we don't vote it in at this meeting, I didn't expect to bring it up again for C23.

Keaton: what Seacord is talking about is that he wants the final version of the paper to be ready to be voted right in.

Svoboda: If we do that work, the paper still may not make it onto the agenda due to there being too many other papers that came in earlier. Is there still a chance for this to get into C23 with the editorial committee?

Keaton: the deadline is one month from tomorrow, so there should be time for people to comment on it. If it doesn't make it, there's going to be another standard coming along.

Svoboda: I'm happy to run that editorial committee, with whoever joins up to do it.

Banham: this proposal implies a very large number of functions given the permutations of types, let alone bit-precise types. How is this going to be implemented?

Svoboda: no bit-precise types with this one, at least. The only things that definitely have to be functions are the constructors. Everything else is a macro, so we don't have to introduce a lot of functions unless the implementation elects to do that.

Banham: that depends on the implementation of the macro. But you mentioned earlier that integer promotions also happen, which adds complexity. I assumed the macros would be using `_Generic` and then dispatch to a macro.

Svoboda: the prototypes do use the `_Generic` keyword to do it, but implementations can do whatever they want.

Ballman: Given the wording issues and the lack of implementation experience, I'd rather we not put this into C23. Adding something this experimental this late in the game with this little information on implementability is a worry. For example, the `typedef` integer types in C may cause interesting problems for people trying to write `_Generic` macros to support this.

Svoboda: FWIW, the prototype uses generics to only distinguish between two- and three-parameter forms, but the core proposal is based on the type. But it's not so much about the type, but it's the sign and signedness. It seems very tractable to support the relatively small number of non-promoted integer types in the language. Does this address your issues?

Aaron: not really; for me, it's mostly about demonstrating that you can implement this without a combinatorial explosion of functions or expression evaluations in the presence of all the different typedefs.

Keaton: we can take it as input and move on for now. The take away is that if it's not voted into C23, that doesn't kill it, it can get into future standards.

Svoboda: we can wordsmith the paper, but there's no way to get implementation experience done in two weeks.

Seacord: suggests we vote on C23 now and if it fails, that tells us how fast you need to turn around the wording changes. If it fails that vote, then you have more time to work on the wording.

Svoboda: was looking for a something along the lines poll with words about "with the current level of implementation experience"

Gilding: I was assuming this would be handled by `__builtin` stuff, but if you can make it portably implementable then there's likely only going to be one implementation everyone converges on. But if it can be done portably with `_Generic`, do we even need to standardize it? I'd personally like it to be there, but it may not be *necessary* to put it there.

Svoboda: the takeaway is that a reference implementation would demonstrate implementability. But that doesn't solve the immediate issue of needing experience before voting it in.

Myers: just because something can be implemented portably doesn't mean we should exclude it from the standard. We have plenty of examples of standard library functions that are implementable

in portable code (`strlen` is an example). It seems reasonable to put things into the standard that are portably implementable so there's a common definition of the behaviour.

Svoboda and Gilding: agree.

Ballman: are you obligated to handle exact-width integer types?

Svoboda: any operation that can produce an unchecked integer type, you need to have a checked integer type. So no need to handle `int48_t` unless some operation can produce an `int48_t` from a 32-bit or 64-bit integer type.

Straw poll: (opinion) Does WG14 support something along the lines of N2868 at the current level of implementation experience in C23?

Result: 7-8-5 (no consensus)

Keaton: this is not discouragement, just that you have more time to deal with it for C2y.

Svoboda: I'll mail wordsmithing changes to Robert and Jens to keep it flowing.

Seacord: why not submit another paper before the deadline? Just because there's not support now doesn't mean we won't vote it in next time?

Keaton: you're free to do that if you'd like.

Svoboda: I don't see a reason to cram if we're not going to get this into C23, so this gives us time to work on the implementation.

Martin U: I don't just miss the implementation experience, it's also the type generic macros that worry me. Before we put more of those into the C library, we should have more of a sound plan for how to get type safe generic programming in C, which type generic macros don't provide enough power.

Svoboda: definitely not going to solve the type generic macros issue for C23.

7.1.2 Ballman, Issue Tracking for C [N 2947]

Keaton: we have discussed this in the past, need volunteers to use tracking systems.

Ballman: in previous years we had the DR list, which served two purposes: issues and problems in the Standard for clarification, and to help users understand. ISO contacted us in 2017 to say we had too many defects and needed to fix them, using a different definition of “defect”. We therefore stopped using DRs due to ISO scrutiny. This has lost clarity on which papers describe defects vs. other issues – new feature changes vs. retroactive clarification of past intent, which affects different Standard modes. The Papers of Interest list tries to do this, but there’s not much on it. There is not enough clarity to implementers about clarifications vs. intentional changes.

It is hard to report issues to WG14 – WG21 has the Core and Library emails, but C requires a paper submission, which is a huge burden. As highlighted by the SG – C++ wants people to report C bugs, but won’t write feature papers. Requiring a heavy process is a barrier to hearing about issues, doesn’t help the Standard, makes it worse not better.

We need ideas in the C23 frame and it appears like we stopped tracking bugs.

Myers: the Papers of Interest list we are adding to doesn't imply "always meant", but "safe to apply" to previous versions. Implementers don't need conditions. This is less useful for retroactive clarity as it mixes two kinds of paper. This list is not a substitute for a defect list because these are not "defects" in old versions.

Ballman: it may be workable to add a column to the list. Mostly want to solve the issue side and clarify for implementers which are issues.

Meneide: I ran into someone writing a paper for a tiny change – wanting to make something into a typedef is so small, but they had to write a paper because we have no process for small changes. Not only is the paper small but hard to write and because of rule changes the author cannot even present it herself. On receiving emails, I make papers or reflector posts, but this makes it hard to propose fixes without a pipeline. We are not capturing fixes we could be getting, and missing a lot of expertise.

Ballman: I didn't consider that the Editor must receive a lot of this too.

Myers: `timespec` is not a minor or uncontroversial issue; needs input from the Austin Group and definitely needs the full paper discussion – this is not a great example.

Ballman: but the user shouldn't have to write it up-front. We need a lightweight entry to the process so that users don't have to start with a full paper. In WG21, a single-paragraph email spawns whole threads of email discussion.

Bhakta: this emphasizes that people think issues are smaller than they really are. The back end of the process should stay the same, discussing full papers – we can provide a front end to users but would need to filter submissions.

Ballman: are omnibus papers OK? Someone puts emails into a DR-like list, Committee discusses a single N-document and filters real issues out to single documents.

Bhakta: this is fine. I want to avoid any change getting in without the Committee looking at it. So when an issue is identified it needs to be made a paper.

Ballman: obviously we have a different resource level in WG21 – issues go on a list, the issues SG goes down it and prioritizes them; someone gets assigned to draft wording; WG21 didn't use N-documents before and uses P-documents now; we don't want to inundate Dan with N-number requests, which is a burden to him and a bottleneck to us. Want tracking of changes to be very clear – but also to pull out as much process as possible to reduce burden.

Bhakta: that sounds exactly like what we want – unacceptable without a numbering system for every issue – so we need a paper for each actual applied change.

Ballman: If proposed wording is in the omnibus, we can use the existing system.

Bhakta: yes, we already do numbered items now.

Seacord: a while back we discussed GitLab issues, which are in use today. Can we use this system for these issues?

Ballman: we can use our own list but it is opaque to the world and to implementers. The list needs to be publicly readable (but not editable). If we can make it public, that's worth exploring and changes a lot. Is anything from ISO blocking this?

Keaton: No. Best if not an N-document. This is allowed by JTC-1 but ISO resists that.

Seacord: not the N-documents, just the issues.

Voutilainen: we use GitHub in WG21 for editorial changes only; we do have Committee input on technical issues. It is an evolving opinion over the last ten years as to what constitutes an “issue” vs a “paper”. This may not scale to a small group. There is no problem with getting a public version out of the data. There is a complex system of issues that maybe GitHub can use, but not clear if implementable – the toolchains are rather arcane. This does *not* result in haphazard adoption of changes, they are heavily scrutinized even at the issue level. Papers are only created for in-depth issues, but all are scrutinized.

Ballman: grouping issues into a timeslot blocked out for issue papers makes for more efficient than unordered work; blocking it into an omnibus helps speed the work along.

Voutilainen: having a way to fix bugs makes it more likely anyone will report them.

Myers: GitLab issues are useful for editorial integrations – the editor needs to watch out for technical content and send it to the Committee. Really only for formatting and text issues. Could imagine a system to automatically extract a paper.

Keaton: is it possible to make a *separate* public issue list?

Uecker: yes. It is a full group and can have multiple repos with their own issue lists. We cannot allow new issues without an account.

Keaton: perfect.

Bhakta: so we’re looking for blocks of time like previous DR-time? I like it.

Seacord: the problem is everyone needs an account, which is managed by GWDG. Can we create a “public” account for commentary? Differentiate public and Committee issues.

Myers: can we number issues starting above the DR range?

Ballman: definitely.

Gustedt: we need an issue tracker; need an easier document system; need a person to manage it all.

Ballman: I plan to write up the process description before taking volunteers to do the work.

ACTION: Aaron Ballman to write up the issue tracking process next-steps and post to the Reflector.

Bhakta: will we also include ISO DRs? I imagine they have to be out of scope.

Keaton: it might kick off a DR but that would require an N-document process.

7.1.1 How to schedule after C23 (continued from the previous meeting)

Keaton: we can insert a PWI for 1 to three years, so we have between 3 and 6 years to publish. The question is fixed or variable schedule. We seem to be leaning towards a fixed schedule. Should do alternating feature/bugfix releases.

Voutilainen: The issue with alternating releases is that it can get awkward with timing. We shouldn't have a *hard* delineation between bug fixes and features. We can’t tell when someone will invent a

feature, and we can’t tell people outside the Committee to hold completed work for a whole version cycle.

Krause: if we alternate, we could have an alternating time for the cycle. e.g., feature releases could take 6 years and bug fixes could take 1-3. 6 years is a long time to wait for `ckdint`!

Bachmann: we should have a short fixed cycle and we should not do alternating cycles. Too many delays with alternation, which are unnecessary if we are disciplined in what we adopt. This also impacts other standards like POSIX and MISRA that have to lag behind the C standard.

Gustedt: agree with what Bachmann and Voutilainen were saying. I think it's important that we tag whether we consider something a bug fix or a feature (the distinction is important), but which release they go into is less important. People outside of the standard have trouble predicting what they can do in 3-6 years. This is a long time to hold things.

Seacord: agrees with everyone else, short fixed schedule (3 years is) and not alternating. The fix for a bug is sometimes a feature, and we shouldn't constrain the solution space by only fixing problems without modifying the existing standard.

Bhakta: I've not heard much new since last time, but I thought we ended up last time with alternating but without constraints. e.g., we can have a bugfix release but it can still have features. It's more about what we're focusing on instead of a constraint. In terms of timing, short and long releases have different benefits. Historically, we've gone long releases, and uptake of the standards has been fairly low. I'm worried that shorter releases will create a larger backlog of unsupported releases. But it's possible that short releases will make people feel more invested or involved and we'll get better adoption. But shorter releases does make features-with-some-invention more dangerous because we may add buggy features at a faster pace.

Myers: this relates to the charter discussion about invention. A key thing about alternation or soft alternation is allowing time for getting additional experience before we build more stuff on top of those features. Once we've got broader experience with a new features across a lot of implementations, that would help us iron the bugs out of the feature to build on top of it. I suggest if something is more of bugfix focus, it might be safer to put in new features that are not related to features in C23 because we might want more implementation experience with the C23 feature.

Gilding: This would enhance user confidence in adopting new versions. We also need to make sure representation of big new features is fair so that people new to the committee don't have their papers sidelined.

Keaton: we will finish this discussion on Friday, and we'll likely have a series of votes on a combination of time periods and alternation vs not. It's possible we can wrap this up on Friday, or we can finish it at the next meeting.

Tuesday

5.5 TS 6010 Provenance next steps (working draft for reference [N 2676])

Sewell: whether to proceed to DIS ballot, and whether to make `intptr_t` mandator. The latter is really Gustedt's part.

Keaton: it is mandatory in POSIX, not in C, so this is motivated by POSIX. C allows capability-based machines.

Bhakta: wasn't this a later paper for the C Standard too? Maybe we should discuss after that. If it applies to C it will apply to the TS anyway.

Sewell: we have clear consensus for the ballot; there has been no more work and no changes to the draft. Procedure?

Keaton: 5 month ballot, allowing two months for translation and three months for balloting itself after the draft is circulated for the national bodies to translate.

Ballman: can this introduce technical issues?

Sewell: does anyone do this?

Keaton: no, but it is on the books.

(no discussion)

Straw poll: (decision) Does WG14 want to move to a DTS ballot for TS 6010?

Result: 18-0-0 (consensus)

Uecker: would like to mention that other languages, especially Rust, are adopting this, so now is a useful time to progress.

ACTION: David Keaton to start the DTS ballot process for TS 6010.

Keaton: editing will begin immediately by ISO editors. There may be wiggle room in the time-frame for important issues.

Sewell: what about the future of the SG? Can it stay together?

Keaton: we should maintain the group as there may be questions arising from editing. Congratulations to Peter!

5.6 Douglas, C2x `fopen("x")` and `fopen("a")` v2 [N 2857]

Douglas: this was last seen in London in 2019, but took a while to deliver. There are two separate wordings to adopt, uncontroversial, and in separate votes. N2357 preceded this and said something similar – dangerous wording can be fixed without a change to intended semantics. `iostream` will build on the `'x'` mode. There are many ways to interpret the current wording, which is a problem for portability; building a lock-file bases semantics on platform behaviour, depending on the "modification" and "usage" definitions of exclusivity. We propose the Standard should imitate POSIX.

Bhakta: on exclusivity – interpretations 2, 3a, 4a don't seem right and should always fail, which reduces the problem set.

Douglas: I see this as implying the wording should be clearer. Normative wording should be unambiguous.

Bhakta: some are non-conforming already, which affects fix direction.

Douglas: `fopen("a")` is much older and there is more risk in an old change. POSIX has the atomicity requirement in the wording already. Very useful, the algorithm for filesystem lock works over NFS/Samba using only append. Originally had something else – C puts the flush in a well-defined place, all implementations except Microsoft already do it (and Microsoft have committed to change theirs). Original wording "to the extent" is now a more explicit failure requirement.

Seacord: as on the Reflector, the issue we hoped to solve and hope that POSIX does, is TOCTOU. The user wants to "create if not exist" without overwriting, and two operations have a race window so we need a single atomic one. There are two problems: firstly, the wording calls for cooperation with an attacker – they won't use `'x'`! Secondly, in the wording "other threads" - the attacker has a process, not a thread – is this what we mean? We need to be clear that it applies.

Douglas: threads are defined, but processes are not, in the C VM. If you don't have the exclusive flag, it will overwrite (with POSIX `open`); to strengthen against attackers would change existing implementations; trying to map what exists right now is maybe not adequate.

Seacord: we learned the first time that this doesn't solve the issue. `'x'` was meant to fix this in C11, but didn't fix TOCTOU there either. This is not achieving the design goal and is critical to fix, especially if POSIX doesn't.

Douglas: proper file handles and sockets in C++ didn't solve TOCTOU there either. We can only give things to the user but can't define what platforms don't, and have to be trailing platform features.

Seacord: what do we tell the user to do to solve the problem? We need to indicate a short term lock, recognized by other operations.

Douglas: POSIX locks are advisory, require cooperation that an attacker can ignore. Need permissions anyway – rare window on setting access, but not against the user's own process. Nobody has anonymous files which would be needed.

Seacord: I don't object but it doesn't solve enough; need a secure directory or something. Doesn't go as far as I want.

Bhakta: Stoughton's second alternative might have addressed this by making it implementation-defined, this improves the Standard, doesn't limit it to a cooperative attacker because the implementation can do more, better compromise.

Douglas: conflating "exclusive" between use-permission and the filesystem operation. When you ask for a unique lockfile, other processed *need* read-access and non-exclusive use. Other times, access needs to block. Suggest two different letters. Our experience with a filesystem DB was lots of exclusive file creation; with files used as IPC buffers, the creator is the leader, populates with content, and then hands it off.

Bhakta: that makes sense but is different. Stoughton's allows this – two meanings are addressed by the implementation. I would prefer the second letter but this may not fix Seacord's case.

Voutilainen: I am confused, does this map directly to `open` modes?

Douglas: yes. Implementations don't implement exclusive access to the file, only the filesystem name. This returns to mapping practice.

Voutilainen: then this should solve TOCTOU because it either *will* create the file or *will* fail. There is a window but it would end in a failure, not a TOCTOU. The wording doesn't depend on the other user using `'x'`, exclusive always avoids it. There is no other mechanism in POSIX that would.

Douglas: so delete the phrase about the other process.

Voutilainen: Seacord has been teaching correctly – this does solve TOCTOU, otherwise your filesystem doesn't work at all (except NFS).

Douglas: NFS v5 will fix this soon, and v4 *can* choose to implement it.

Voutilainen: if we teach this to users, we will tell them about `'x'` semantics, they will learn via `open` semantics – it isn't surprising and there is no need for weasel-ing.

Douglas: the append changes are mostly Seacord's, defining atomicity on flush or write. This will also fix `fopen_s`. All major implementations are compatible, POSIX `O_EXCL` and Microsoft.

Gustedt: "must" wording should be "shall".

Voutilainen: problem with reading "atomicity" into the definition of append – no wording suggesting process synchronization. Seems vague but good if it works in practice.

Douglas: this is needed for logfiles not to have torn writes. It is the source of the ZFS write-halt, which is a pathological case; can't fix the semantics though.

Bhakta: still not OK with the wording not covering this – can't count on implementations doing so. I don't know how to word it, do we know any implementations that would change?

Douglas: I can't comment on what isn't listed, but all the ones I know use `O_APPEND`. Nobody from POSIX could think of one. You're not wrong that adding atomicity here *may* break, no existing implementation provides exclusive *use* – so it wouldn't break anyone to remove this. Adding a new letter may be harder for POSIX to do.

Seacord: exclusive *use* was never the original intent. We always understood the language was confused. Annex K defers to the platform anyway, don't think we need to try to do that. For `fopen("a")` all implementations also do the same; `fopen_s` is Microsoft-only.

Keaton: sorry about the mailing list issue – it removes users on a bounce. No malice was intended.

(no vote)

5.7 Köppe, Comma omission and comma deletion [N 2856]

Köppe: this originates in 2017, to distinguish when the ellipsis is empty. Was well-received by C++, who found corner cases and fixed up subtleties in wording and interaction with # and ##. Wanted to just take the C++20 wording, but Gustedt has proposed a simplification; Tong has identified issues with it.

Gilding: this *needs* to be adopted to maintain preprocessor compatibility. Maybe even better to adopt the feature in an incompatible state than to leave unadopted and risk preprocessor divergence as we saw with C99 and the original introduction of the ellipsis/__VA_ARGS__.

Bhakta: I would rather have to deal with complicated `#ifdef` than silent incompatibility. We should get it right on the first attempt.

Köppe: I expect the useful cases not to be contentious.

Tong: one non-corner case is that empty macros are empty for __VA_OPT__ in C++, but not in this wording. There are also many other cases that would be silently different. I don't know the history of the "pseudo-parameter", but the C wording doesn't require atomicity of the paired parentheses, which is very important and can hit an expanded `)`. Moving forward with the different wording is a bad idea. I don't like the design changes and silent changes are harmful to fix.

Myers: it is very important to ensure compatibility with C++ - we noted the issues before.

Ballman: agree that divergence is a problem, but having the feature in C++ will just solidify it anyway.

Köppe: I do have an unpublished draft of just porting the C++ wording across, not sure of any issues.

Tong: so the proposed C wording does something different that will lead to confusion; if details matter then voting on this isn't helpful.

Gustedt: please explain the difference.

Tong: in C++ __VA_OPT__ is an atomic entity, parsed as a complete unit like a parameter, so it will work with # on expansion. C treats it during rescan, non-atomic and not-expanded. It is not empty before rescan when considering further expansion.

Köppe: about wanting the C++ semantics, vs preferring what exact wording?

Gilding: I don't like diverging in the implementation and would not expect anyone to actually implement this as-written. Implementers will not provide two separate semantics for their preprocessors in practice.

Bhakta: in that case what is the use case for divergent wording at all?

Köppe: we found the C++ wording too complex and difficult to formulate. Changing the meaning was not intended. Happy to rework it to get the C++ semantics.

Keaton: so we have no arguments in favour of diverging.

Köppe: I will bring a new paper with exact C++ semantics.

(no vote)

5.8 Bachmann, Make pointer type casting useful without negatively impacting performance - updates n2484 [N 2658]

Bachmann: the sticky nature of effective type allows efficient access but disallows access by other types, even non-overlapping; doesn't allow partitioned memory areas; we try here to limit the effect of aliasing and limit to immediate cast. Half of the functions in `<string.h>` access memory

through a larger size. This is important for `memcpy`, `memcmp`, `memmove` etc. that cannot be implemented efficiently in conforming C (even `malloc`). Users find immediate-casts clear to read; there is much existing practice so this is supported by the Charter. A small hole in the type system allows for useful work despite wanting a safer language.

Gilding: I consider this a safety improvement vs. the alternative, which is to have this practice be undefined.

Bachmann: we only allow one level, as seen in the "valid" example line. Only when the operand is immediate; Myers observes that this term is not well-defined so we should define it in the grammar; I dislike Myers's new syntax and want to reuse practice – syntax only discusses pointers and should allow brackets and casting from `void`, but not use as function arguments which are not immediate dereferences.

Gilding: so the converted "immediate" pointer is not assignable and therefore not reusable in any other context?

Bachmann: yes, we don't intent to introduce any tracking.

Gustedt: don't think we need "immediate", the operand is direct. Existing terminology needs to apply correctly.

Sebor: don't follow the rationale – these seem like highly specialized use cases. The paper mentions `-fno-strict-aliasing` – assume this is equivalent?

Bachmann: no, because that option is global, this is tightly bound to local usage.

Sebor: have you tried the impact on optimizers?

Bachmann: we have seen local casts; there are no counterexamples; the GCC mailing list pointed out that we cannot rely on non-standard practices to behave as expected. In the absence of the casts, nothing changes for the optimizer.

Keaton: there is no aliasing problem because this pointer can only be used as the operand of `*`. It's not assignable, to create an alias it would need to be.

Sebor: I advise against adopting this without implementation experience.

Bachmann: without the optimizer enabled this *is* implemented – the optimizer does more than this.

Sebor: so compilers set to use `-O0` conform already.

Bachmann: that option does too much.

Sebor: introducing standard changes introduces issues for optimization.

Bachmann: it is not adequate to be afraid. This is a radical position without a plausible issue.

Keaton: I do not believe this would affect the optimizer.

Steenberg: this is a good proposal and good for performance. Without the change the behaviour is undefined, user already do this in real life, and optimizers are allowed to do weird things, making the rules clearer would make more room for compilers to work. Really useful to allow more aggressive optimizations, more mainstream compilers would benefit. Not allowing alias on assignment is surprising.

Myers: this is not equivalent to global `-fno-strict-aliasing`, it treats immediate casts as having `[[gnu::may_alias]]`, or the closest equivalent since an attribute would be ignorable.

Uecker: it would only affect optimization for expressions where you have this, so it is benign. It's not hard to benchmark by replacing `memcpy` etc., but unclear what result is acceptable. The Linux kernel requested this behaviour, they want the compiler to do the obvious right thing. Alias analysis just hurts unclear code. I very much prefer this established syntax; better optimization provided than alternatives by constraining the impact to immediate operands.

Bhakta: I like the proposal but do see Sebor's point. Alternatives imply vague experience but unclear how direct.

Gustedt: what would experience mean? This is UB, we are adding a permitted action. No compiler can make a valid change yet. What constitutes the measured outcome? The argument is vague because of the lack of implementation.

Sebor: within GCC this is a broad-impact feature. Patch it then benchmark the impact. If it's not positive we can analyse what the issues are; right now we have no data.

Gustedt: this doesn't answer the question of what an implementation is – do we need a broken compiler?

Sebor: it is a patch with benchmark data.

Gilding: I think this is inherently impossible to measure because we can only change a misbehaviour.

Bachmann: I do not have the resources to test this, it is only my opinion. Myers's concerns need to be addressed. This will not be the end but I do solicit further support.

Straw poll: (opinion) Does WG14 want to see something along the lines of n2688 in a future version of C?

Result: 11-0-6 (consensus to continue)

5.9 Johnson, Unicode Length Modifiers [N 2983]

Johnson: minor updates based on Myers's feedback. Main change is the capital and lowercase `u` – lowercase conflicts with the specifier, does anyone know of the capital being in use?

Bhakta: "no" as a length modifier, but yes in general. We use it for "unblocked data". No issue with capital `U` as a length modifier.

Krause: "...shall be provided by" introduces problems – we need "as-if" wording and no internal state. Optional functions and don't want an internal object.

Johnson: Meneide also mentioned this in the context of the conversion functions paper.

Bhakta: w.r.t "code point" as a Unicode term – this can refer to other encodings, don't want to bind the term to Unicode.

Johnson: `char16_t` and `char32_t` are adopted and defined for Unicode only already.

Honermann: "code point" is in ISO/IEC 10646, which is a normative reference.

Johnson: we discussed whether to include this and should update with that.

Keaton: since it's a normative reference, it's as-if it was included.

Bhakta: so this is making a difference in the C Standard.

Honermann: you may want UCS-scalar and to exclude surrogates. That is a more fitting term that describes the aim better.

Johnson: was unsure if this was obsolete and how to fit it together.

Bhakta: this would no longer affect me.

Gustedt: I don't think it makes sense to remove the mention of Unicode – it's the central reference for what a character means, everything maps to it; alternative encodings map onto it, so this *should* refer to Unicode.

Bhakta: using UCS-scalar instead?

Honermann: don't use it to discuss the output, distinguish the output type.

Bhakta: in 8c, "`int` argument"?

Johnson: cleaned up that section to try to make the types clearer, is now `u32`. Problems in 12 have been fixed in new version.

Bhakta: do we have experience?

Johnson: privately, yes. The Clang patch has not been accepted yet, has many dependencies.

Honermann: the conversion from UTF-8 to the wide execution character set is not correct, should be locale-dependent. "Character set" doesn't refer to locale-dependent forms, the terms refer to the encoding of literals.

Krause: w.r.t experience, do we have experience for the length modifier? For printing UTF-8 we have loads of experience, plenty printing UTF-16, `wchar_t`, etc.

Johnson: I agree, not much here is new, it's just a repackaging.

Bhakta: I was asking not just for `u8` but generally implementation experience with length modifiers.

Johnson: I have done it myself, it isn't hard to do – other non-Clang compilers may have more difficulty.

Straw poll: (opinion) Does WG14 want to see something along the lines of n2983 in C23?

Result: 6-2-5 (consensus)

Keaton: there is a high bar for the next version.

Johnson: Honermann's point is the biggest part, plus UCS-scalar.

Bhakta: do this, but not for C23.

Banham: similar to Bhakta, this feels too rushed. I am concerned by the conversions; `char8_t` is similar to characters, plain strings with UTF-8 encoding, with understanding deferred to the device.

Normal `printf` should use multibyte, not wide characters. We can ignore wide unless it converts to an internal representation.

5.10 наб, `nsec_t && timespec::tv_nsec` [N 2878]

Meneide: this has a lot of ABI impacts, but not too bad because this proposes a typedef, that allows implementations to avoid a conversion between kernel object and user object. Currently we have `time_t` for seconds and `long` for nanoseconds; the fixed type is a problem when `long` is the wrong type, such as when the kernel mismatches the word size. At compile time the implementation should know better. A typedef would have identical range requirements, but be implementation-defined. This is harmless because implementations can keep the existing type, including `long`, but others have the freedom to change it to fit new requirements.

Krause: don't oppose implementer freedom, but don't think naming the type is necessary. Avoid discussion of what you can do with it.

Meneide: we wanted precedent for specifying structures without specifying the member type, but couldn't find it.

Bhakta: does any code depend on it being fixed to `long`? Would this impact `_Generic` or overload resolution?

Meneide: it is a concern; we didn't have numbers. The previous impediment wasn't that, but rather than implementations would stop conforming.

Bhakta: there is a generalized dependency on `long`.

Myers: `printf` would most likely depend on it, for the `long` specifier. The Linux bug was fixed to ignore the high-32 bits in the 32-bit ABI, and will now correctly interop with user code, so the fix is no longer needed. It is not clear that a typedef is needed now. We would do this if it was new, but this need collaboration with the Austin Group.

Meneide: I *think* they're amenable but will contact them anyway.

Bachmann: I think the type should remain signed for arithmetic usage, so that unsigned types cannot affect promotion.

Meneide: we can easily do that.

Gustedt: I think changing a POSIX-inherited structure needs POSIX input. The main issue is what this fixes, the minor issue is unlikely to impact anything, the major issue affects something nobody uses – this is a poor use of energy.

Meneide: we do have a user who was willing to escalate it. Maybe an implementation-defined type without a name? Will discuss with POSIX. The fact a user reached out is significant.

Bhakta: I don't want to go forward, but also not to tell the author this wasn't important – the author must be respected.

Straw poll: (opinion) Does WG14 want something along the lines of n2878 in C23?

Result: 3-3-9 (no consensus for C23)

Meneide: I will take this to the Austin Group.

Gustedt: no time to coordinate in four weeks.

Meneide: can we bring this for C2y?

Bhakta: no objection. Papers are allowed from everyone.

Wednesday

5.11 Meneide, Restartable Functions for Efficient Character Conversions (r7) [N 2966]

Meneide: received multiple email feedbacks on wording; ensures identical behaviour for double-indirections; clearing matches existing practice. We do provide operational definitions of “code point” and “code unit”, given yesterday we may move these – changes to clarify intent needed, much feedback from Tong; p6 is the part specifying the link between the input type and expected encoding – also needs to move to a top-level area for more general applicability. Question about a sequence of code units representing a single code point – before, reading one and storing the state was OK; now this is *not* intended to be well-defined for the UTF-8/16/32 encodings – full-complete is intended, or to return incomplete and not touch the output. Predictably consuming all input avoids implementation divergence; this only applies to the UTFs and their implementation-defined equivalents. Forces consistency, can define other encodings but they mustn’t be the same. Zero causing a reset formalizes existing and reference implementations.

Voutilainen: was the naming scheme discussed previously? Why not just the UTF-encoding names?

Meneide: this fits the precedent set by existing functions.

Meneide: we allow space for flexible intermediate formats, encouraging complete reads. Received a lot of comments on the exact wording, especially nulls and zeroes; added `restrict` to the API.

Bhakta: have any comments been wrong?

Meneide: no, all were correct.

Banham: the Standard terminology for “execution character set” is unclear; the extended set covering characters and wide characters is vague when not Unicode; is there an opportunity to clarify the distinct character sets?

Meneide: previous papers addressed this, n2728 pinned down `char16_t` and `char32_t` as Unicode.

Banham: these are still very tied to the implementation decision about the character set. A program reading in can only assume data conforms.

Meneide: have to agree with the implementation or use out-of-band data to mark. A lot of databases still use a system locale so these allow a way out to portable Unicode. We need something in the future but would be impractical to provide here.

Banham: contrast ICU: this is not that, but is it general enough? `char8_t` being fixed to Unicode is good, but `char16_t` and `char32_t` are not fixed.

Meneide: we changed these to be UTF-16/UTF-32. There was no contradictory experience, confined to the `mb` and `wc` functions – we don’t need to worry about the `cN` types. The

implementation this is based on has additional standardizable functionality, but there is less practice/adoption of untyped conversions.

Honermann: concerned that defining “code unit” here conflicts with ISO/IEC 10646. I would prefer an alternative abstract term.

Meneide: currently we draw a distinction between “code point” and “Unicode code point”; more references to the general concept than to just the Unicode concept.

Honermann: I agree that the general term fits, but unsure about potential conflict.

Meneide: if we fixed, eg. “abstract code point”, it would be symmetric with “abstract character”.

Bhakta: I disagree – “code point” is widely used and understood beyond ISO/IEC 10646. It wasn’t meant to be a general definition only for its own purposes. What’s here is normal, 60 years of usage, non-specific and widely understood.

Sebor: is the implementation shipping?

Meneide: yes, we have customers and commercial product support. Punycode is included in a shipping product, and has articles in the Journal of Open Source Software.

Sebor: great. That constrains changes we can make. The order of arguments differs from existing APIs – this is a usability issue. About 50 APIs with odd names; are there alternatives? Can these be wrapped in virtual functions? Have you considered simplifying changes?

Meneide: trying to mimic functions in the Library. Prefix+encoding+restartable, then suffix. Prefix had to change for `mc`/`mb`, multi-*char* vs multi-*byte*. Single-value functions with a different meaning. The names are cryptic but spelling the out would hit the external identifier limit. We didn’t have a solution beyond following Library practice.

Sebor: hence wanting to encapsulate them as pointers. In C++, facets have input and output in a named structure.

Meneide: we didn’t look at those, but did find precedent for functions only. w.r.t the ordering, we wanted to use `static` for the buffer sizes, which unfortunately doesn’t work with indirect pointers. We imitate existing practice in `iconv`. Putting the multibyte string last is where the Standard always puts it. It’s hard to accidentally get the wrong thing because these are not `void` pointers and there is no implicit conversion.

Krause: since these are not VLAs, we don’t need the size before. C26 might improve this with usable sizes; why not make them generic and type-determined?

Honermann: that type information doesn’t exist.

Meneide: we do have that in our library, but you can’t detect the type of `NULL`, which hurts usability. We have it working decently well, but did not propose it because it was original invention. You do need to be able to form pointers to these functions, so need the names anyway. Have been trying to cut down the number of functions. We found the exploding API problem in the library, avoided it with generic functions, avoiding 2500 APIs – but not enough usage in this proposal.

Honermann: we need to ensure character types are distinct. Some functions never use state, why is it there?

Meneide: it keeps the interface the same across all APIs, makes it generic, easier to write macros over the top of these. Will add explanation.

Bhakta: I oppose Sebor: I like the design, based on existing APIs around since C95? Naming mechanism and interface is totally clear to experienced users, following `iconv` is great, would not want to change this.

Sebor: do these integrate usefully with `stdio`? I believe GNU libc uses a virtual function API internally.

Meneide: yes but not trying to make that change right now.

Krause: can we have wording to guarantee `NULL` in the unused state slot is safe?

Meneide: I think that's implicit – passing `NULL` already means use local state where needed.

(no vote)

5.12 Meneide, Enhanced Enumerations (r4) [N 2963]

Meneide: a lot of edit requests. This continues Pygott's work. Some more changes since published version. Defining a type for enums fixes the problem of `int`-typed constants having non-portable range. Significant work required to get parsing to work, limitations of C grammar. Declaration/definition combinations prohibited to prevent ambiguity; compatibility requires explicit compatibility. Tweaks made to wording of underlying types. Specify that enum values must be in the range of the underlying type without conversion, with an exception for `bool` underlying type, which follows existing practice.

Seacord: in the examples you have e.g. signed integer literals, assigning to unsigned enums – is this conversion?

Meneide: the *value* must exist in the range without conversion, not that no conversion can happen, just that it must fit. This prevents the use of out-of-range values and overflow/wraparound. Added wording for wrapping which is a definite constraint violation, following existing practice. "Longest sequence" breaks ambiguity, and the constraint against using in declarations without an enumerator list – so no forward declaration in parameter lists or function returns, bit fields, typedefs, etc. You *can* do a simple forward-declaration and then use the name separately, the wording may need adjustment. Enumerators now have the same type as the underlying type without divergence. Bit-precise integers are excluded; this was never intended to work in Clang.

Seacord: what is happening with arithmetic in paragraph 14? The "overflow" term may be a better fit with Svoboda's concept of infinite precision. It wouldn't wrap either – it would promote then truncate on the assignment.

Meneide: so `USHRT_MAX + 1` is too large for an `unsigned short` without conversion. We can workshop the language.

Pygott: thank you for finishing this work. I was stuck. Thank you for resolving it.

Meneide: this feature and the next one receive a lot of user requests.

(no vote)

5.13 Meneide, Improved Enumerations (r1) [N 2964]

Meneide: also many fixes based on feedback. Same derivation from Pygott's work. Implementations struggle and do odd things when going out of range – supposedly a constraint violation but too much practice means a lot of illegal extant code. Enums can be non-portable depending on the width of `int`. We can provide a standard value behaviour for this code while still allowing implementations freedom. Some compilers don't even produce internally logical values. So no longer `int`, but instead "some type" – we can't mandate a concrete type, can ensure the value is not lost at least. Naming the "enumeration member type" will become useful to other places.

Krause: we do need to mention bit-precise types here too, may have a long-term effect to exclude them right away as can't be fixed later.

Meneide: the authors at Clang/Intel didn't want to use them that way.

Krause: it might be useful to embedded.

Meneide: since it is totally implementation-defined, removing that is fine.

Ballman: this means enumerators sometimes promote and sometimes not. We really need that to be up-front.

Krause: if you can only choose a type bigger than `int`, there is no promotion issue.

Meneide: dubious – documents existing practice, but the practice is a mess and shouldn't be taken as a guide. The rules here are too complicated – practice shows we need a better option. It is reasonable for enhanced enumerations to be bit-precise but questionable to allow implicitly. Would prefer to only proceed with enhanced enumerations.

(vote deferred)

5.14 Gustedt, Remove `ATOMIC_VAR_INIT` v2 [N 2886]

Gustedt: nobody needs this feature. We noticed other corner cases for atomic initialization at the same time. Signal handlers are allowed to use lock-free atomics; regular initialization not protected the way atomic initialization is. Should make explicit that touching bytes, allocated objects etc., has data races. Besides this removal, Variant 1 is non-normative. Variant 2 adds a normative sentence with UB for signal handler usage.

Krause: what is "modification" of an object in indeterminate state? Initialization modifies an object, is it UB to have a local atomic inside a signal handler? Maybe to be shared with others? So it gets initialized but that is a modification?

Gustedt: Variant 3 is Boehm's observation that it is bad to allow for *non*-lock-free and force `atomic_init`. This is a significant normative change. Variant 4 would be for all atomic types and a major normative change. `atomic_init` explicitly imposes an effective type.

Gilding: is there precedent for Recommended Practice aimed at the user rather than the implementer?

Gustedt: yes. We have both.

Bhakta: what does implementation experience indicate? Would anyone be broken?

Gustedt: not implementations, but user code would be very restricted. All known implementations allow uncontested assignment.

Ballman: `rand` is the only example of RP aimed at the user.

Gustedt: generally we formulate it as examples.

Bhakta: Variants 1 and 2 would cause no problems. Variants 3 and 4 have implications for user code.

Gustedt: there is clear practice that `=` is usable for initialization.

Straw poll: (decision) Does WG14 want to integrate Variant 1 of the change specified in N2886 into C23?

Result: 14-2-2 (adopted)

Straw poll: (decision) Does WG14 want to exchange Variant 1 with Variant 2 of the change specified in N2886 in C23?

Result: 9-3-6 (adopted)

Straw poll: (decision) Does WG14 want to integrate Change 3.5 in N2886 into C23?

Result: 8-1-9 (adopted)

Straw poll: (decision) Does WG14 want to integrate the Recommended Practice in Change 3.6 of N2886 into C23?

Result: 2-5-9 (no consensus)

5.15 Gustedt, Require exact-width integer type interfaces v2 [N 2888]

Gustedt: this polled favourably before. Change 3.1 is already integrated. This is the last part of the `intmax_t` replacement.

Straw poll: (decision) Does WG14 want to integrate changes 3.2, 3.3 and 3.4 from N2888 into C23?

Result: 12-1-4 (adopted)

Gustedt: this has extremely limited impact and mostly affects types not yet provided. Width macros allow detection and guarding in preprocessor-expressions.

Bhakta: OK in general but unsure about combination with next paper. Forcing `intptr_t` may be difficult.

Gustedt: problem was raised in the liaison SG. This paper is much more important – the lack of ability to define extended integer types is problematic. All major implementations have the types, but require extension names to use. The other paper is portability.

Bhakta: agreed, this is more important.

5.16 Gustedt, Pointers and integer types [N 2889]

Keaton: clarifying that this modifies TS 6010, which is out for ballot.

Gustedt: right now the type is optional. Recent discussion found only two architectures without the type, both stuck by the `intmax_t` problem. Was pointed out by Tong that underlying assembly might have hidden bits, playing a provenance role; back-conversion from an integer to a pointer is not possible from non-provenant bit patterns. User code circumvented it anyway using punning. Having the integer type does *not* imply consistent arithmetic with pointer arithmetic. TS 6010 impact is that ordering must respect ordering across the type, so `<` still works.

Bhakta: the case with extra bits is not obscure, bits matter, the union pattern preserves the bits whereas forcing a cast loses them, which makes it worse.

Gustedt: a type provided must include hidden bits?

Bhakta: provenance bits don't form part of the main pointer and are likely to lose values. This will make bad practice more common. I see unions as explicit preservation of the extra bits.

Bachmann: we need `intptr_t` because otherwise we cannot use `realloc`. There is no other valid way to test if the pointer was moved. There is no way around the need for this.

Gilding: the arithmetic problem already exists so this is not a big deal. Many machines have extra bits for other purposes such as GC tags or NAT. Is the relation requirement true in general?

Gustedt: not sure, limited to the TS. As far as I know implementations can choose to give result values an ordering, right now the ordering is only defined for objects within the same array, hard to see how that can go wrong.

Myers: as expressed in Reflector 20542, I am not convinced, this only affects very-wide-pointer architectures, which are the ones where the integer representation isn't useful. Won't have arithmetic properties and manipulation is impractical. `realloc` etc. is dubiously useful. All the things we use this *for* are only useful where it currently works anyway, and not on other platforms. Might be better to allow bit-precise integers larger than `int` for these platforms to avoid extended integer types.

Gustedt: I do think there are marginal use cases even when arithmetic doesn't work. Unsure about bit-precise integers – wouldn't want for some architectures but opening the option makes sense.

Bhakta: for people who have this it's no issue. For people who don't this is a *high* implementation burden for only marginal benefit. Would a `WANT` macro make more sense?

Gustedt: you can already feature-test with `INTPTR_MAX`.

Keaton: personally, experiments are because of security, especially GPU memory and capability-based architectures. I prefer to add to the TS, not to C, and give a chance for experimentation.

Bachmann: `realloc` is clearly a benefit.

Banham: it's only true if it requires 128-bit arithmetic if that is the pointer size. Two changes are implied: an optional feature made mandatory, which is separate from allowing it to be wider than `intmax_t`.

Myers: w.r.t capability-based architectures, some (Cherry) don't support standard C, pointers are 129-bit and side bits convey validity. We need more investigation on how these *can* be supported before putting it in C23.

Gustedt: I support Keaton's proposal for the TS. Need to invent a mechanism to make setting this portable.

Straw poll: (decision) Does WG14 want to integrate changes 3.1, 3.2 and 3.3 from N2889 into C23?

Result: 4-7-7 (no consensus)

Straw poll: (decision) Does WG14 want to delay the DTS ballot for TS 6010 and integrate changes 3.1, 3.2, 3.3 from N2998 into TS 6010 before balloting it?

Result: 8-3-7 (adopted)

ACTION: David Keaton to notify Peter Sewell that we are delaying the ballot to make the change in n2889.

5.13 Meneide, Improved Enumerations (r1) [N 2964] (continued)

Straw poll: (opinion) Does WG14 want something along the lines of N2964 in a version of C after C23?

Result: 8-5-4 (consensus to continue)

Myers: prefer to only vote in Enhanced Enumerations.

Gustedt: impression is also that this is too complex – adding procedure is bad, prefer to force the type. Not opposed to adding this later.

Bachmann: agree with Gustedt, stacking rules on rules, C programmers are often not aware of the rules, still fixing `bool` 20 years later. Rule changes need to be natural or they make the language less reliable.

Seacord: capturing behaviour in the Standard tells users what they can expect, it is better to say than to not.

Bhakta: typed enums make everything explicit – at best, this encourages implicit usage, making the problem weirder. Prefer to encourage clearer and readable explicit code rather than this.

Meneide: so we can choose to standardize practice, but would like users to choose Enhanced Enumerations. Implementations all either try to do this, or mis-compile. The code is already non-portable in circulation. Saying we want users to rewrite old code only helps what will be redone. This does preserve values, even though it doesn't settle types. This takes mis-compiled code and makes it work without needing to fall back to `#define`. Too much code would need fixes and only in C23; this forms a basis of guidance for what should happen and at least keeps enum values portable, giving worthwhile behaviour to old code.

Gustedt: if it exists, the different things that implementations do changes code semantically. We should make clearer that values not fitting is a constraint violation.

Keaton: the vote was for a *non*-C23 version, so we have sentiment to proceed.

Thursday

5.17 Uecker, Indeterminate Values and Trap Representations (updates N2772) [N 2861]

Uecker: this was received very positively last summer. Clarifying wording without intending semantic changes, and preserves assumed semantics exactly, originating with the Memory Object Model SG, partially from the UBSG. Added further explicitness, removed “trap” wording, focuses the wording on object representations over values, “unspecified” where the focus is on values.

Krause: “trap representation” is an established concept – is it worth changing it? Not zero-cost.

Gustedt: the change has an effect on `atomic_init` – think the editor may change because it’s common to the merge of independent papers. The “trap” change is good, gives users wrong ideas.

Gilding: I like removing “trap” – it doesn’t imply that and an actual trap doesn’t require the word.

Seacord: this will probably improve teachability a lot, this is something difficult to communicate to students.

Myers: C++ doesn’t have “trap representation”, may not have an equivalent term, but if there is we should be consistent.

Uecker: “indeterminate value” is used differently in C++. The merging issue seems editorial.

Keaton: “trap representation” was for Annex H, which is no longer there.

Myers: what about subsequent wording? “Indeterminate” in 7.21.2 changes meaning; before the change it implies a total overwrite of subsequent bytes of file, new wording doesn’t allow that. Not sure this affects any implementations and is probably not harmful.

Wiedijk: so this is terminology only and doesn’t affect “wobbliness”?

Uecker: this has no impact on whether wobbly values exist at all. The current wording doesn’t support them.

Wiedijk: this steers the discussion in a way, avoiding “trap” appearing in future.

Ballman: C++ does use “indeterminate” for the same concept as C – inconsistent use might be confusing.

Uecker: disagree, in C++ indeterminate values can be copied around and preserve the indeterminate property. In C, reading is UB on the value-conversion, so never reaches the value. It is helpful to have a different name.

Ballman: not sure I agree but it’s hard to say. C++ objects can have indeterminate values, and reading them is UB, *with exceptions*. Representation and value are tightly bound; the intent is not to read the value of an uninitialized object, which is the same as C.

Uecker: my understanding is that in C++ this can be copied to a target which becomes indeterminate itself.

Ballman: the exceptions are only for byte/`char` reads, so not sure that’s true.

Gilding: the change to streams seems plausible, but a definite improvement, would not have intended the old behaviour to be conforming.

Bhakta: I kinda agree but see how a simple implementation could change the following content, even if not multibyte. But I don't object.

Straw poll: (decision) Does WG14 want to adopt n2861 as-is into C23?

Result: 13-1-3 (adopted)

5.18 Uecker, Improved Rules for Tag Compatibility (updates N2366) [N 2863]

Uecker: this is an old proposal from London. Obscure rules mean two tags can be compatible *outside* the TU, but not within it! We allow making them compatible and also redeclarations. This makes the language more logical, allows improved generic programming with macro-templates – no need to centralize declarations any more. Can now write sum and produce types inline, which is user-friendly. Changes to incomplete types were removed as too complex. This has been tested but not shared. Wording changes mostly just allow redeclaration to work as expected. There is a subtle change to when types become complete. Gustedt criticised change to types without a tag, which are now compatible always – incompatible would be more conservative, but no impact outside of weird corner cases.

Gilding: does this affect any correct code?

Uecker: it is possible to create a `_Generic` which distinguished two untagged types named by `typedef`, which would now break.

Myers: like the idea but there are issues in the wording.

Ballman: I do like this a lot, but "compatible" and "same" handling is unclear – we have type multisets, could run into that. Users can be surprised that `int` and `signed int` are the same, this would expose that.

Uecker: do we really only use compatibility? This is already exposed across TUs.

Ballman: want to see that users aren't confused by alternate spellings. Have seen user confusion.

Uecker: this isn't worse. I hadn't encountered the issue of typedefs of the same tagless structure becoming compatible when they were not before.

Bachmann: so are types with different tags compatible?

Uecker: no. Never. In the example the tag name is the same.

Bachmann: the member name is different in the example.

Uecker: this doesn't change the existing rules.

Sebor: compatibility rules affect LTO. I don't see an obvious problem but GCC did have some issues here. Try to build a non-trivial LTO project to test. The test suite doesn't serve well here.

Uecker: the hardest part was breaking potential optimizations because of strict aliasing, it isn't broken in my own GCC, but will test more.

Keaton: does this affect C to C++ portability?

Uecker: if you have a redeclaration in the same header, I wouldn't expect it to work in C++, but there isn't a plausible use case. C++ is already quite different and has no tag name space.

Ballman: I don't think compatibility is the problem – redefinition in the same scope is, but C++ has more scopes, namespace and so on. Might be surprising.

Straw poll: (opinion) Does WG14 want something along the lines of N2863 in C23?

Result: 8-3-8 (consensus to continue)

Voutilainen: I do think this is an incompatibility, but don't see it as a problem.

5.19 Uecker, Safer Flexible Array Members [N 2905]

Uecker: this is another follow-up to N2660's general array improvements. This makes structures with flexible array members incomplete. This is a breaking change, but a good one, require a diagnostic and confirm that usages are intentional. Expect no change to compiler behaviour in practice except for the warning. We continue to allow `sizeof` and intend for future direction to deprecate it. The proposed macro takes padding into account, which cannot be done portably.

Ballman: the Future Direction – can we put that in the expression section close to `sizeof` and mention `alignof` as well?

Sebor: seem like breaking changes; I understand the problem but don't see as uncommon. Obsolescent features get diagnosed and prompt the user to change to something, but there is idiomatic usage here, switching introduces risk.

Uecker: that's a judgment call, maybe not strong enough. `sizeof` is wrong, you need `offsetof`.

Sebor: `sizeof` only overestimates. I dislike the constraint violation on assignment – the risk of unexpected misuse is present but a breaking change, which is a dealbreaker.

Uecker: the breaking change is what was asked for; remember this only requires a diagnostic. Unclear if there is a good alternative, I would expect a warning here.

Gilding: `sizeof` is correct. While the idiom is well-known and recognizable and there is risk in change, I definitely want the constraint violation. All existing code doing this is wrong. Use an element, or use `memcpy`.

Bhakta: agree about `sizeof`, but do not agree that the constraint violation is right. There are plenty of valid uses for this assignment.

Banham: so `offsetof` dates back to the use of an array of size `[1]` to simulate the flexible member. I consider it incorrect to use with `[]`, you should use `sizeof`. I am baffled by the nature of the macro and the need for the member designator.

Uecker: it would allow a macro implementation instead of compiler magic.

Banham: not convinced that a macro that expands to a common idiom is useful, especially not if it uses an obsolescent feature. If you do want the array part to be separable from assignment, the header members should be a separate type. I understand this may imply changes but it doesn't seem correct as-is.

Straw poll: (opinion) Does WG14 want something along the lines of making structures with flexible array members incomplete types, except for use in `sizeof`, in C23?

Result: 6-5-7 (no clear direction)

Straw poll: (opinion) Does WG14 want something along the lines of Change 2 in N2905 in C23?

Result: 4-4-10 (no consensus)

5.20 Uecker, Wording Change for Variably-Modified Types [N 2907]

Uecker: this is a follow-up to making variably-modified types mandatory, solely to clarify.

Myers: can we have “is mandatory” in a footnote? Sounds like a requirement.

Keaton: this is not allowed. The same for NOTEs: they can’t include a requirement.

Sebor: maybe state that parameters are *not* conditional, which doesn’t add a requirement.

Gilding: “remains optional”?

Bachmann: just make it normative, then.

Ballman: can we say “variable”? There are also typos.

(Martin Uecker edited over the break and brought back as N2992)

Straw poll: (decision) Does WG14 want to adopt n2992 as-is into C23?

Result: 16-0-3 (adopted)

5.21 Uecker, C23 Atomics: Proposed Wording Changes (updates N2771) [N 2909]

Uecker: there is user confusion over a qualifier that doesn’t behave like the others. No control is provided over representation. ABI is chaotic and has no consensus. Genericity doesn’t work. We have two options: making it a true qualifier can result in a different type from the specifier; or a new qualifier and reuse `atomic` for the specifier.

Gustedt: I like Alternative C the most; guarantees compatibility with C++, breaks minimal code. I dislike the name `_Lock`, it means the wrong thing.

Myers: I like Alternative C the best – versions that change the meaning of a keyword are a really bad idea for usability. Removing the qualifier without deprecation is also a bad idea. I don’t think we need to deprecate it, we don’t *need* multiple syntaxes, but they don’t hurt. Lock-qualified vs. unqualified means conversions become valid from non-lock-free – I would prefer to see implementation experience for atomic access to non-atomic objects.

Gilding: this conversation is guided most heavily by platforms like x86 where the distinction is the least important. Users are surprised most by changing the specifier rather than the qualifier, again because the distinction is weakest in the places where the feature sees the most use. We have the least experience on the platforms where it is most meaningful. What is the impact of embedded locks?

Myers: Microsoft have these but they haven’t provided them for their C implementation.

Krause: we cannot get rid of the specifier because of the compatibility story, and because it is still safe to use embedded locks.

Sebor: compatibility has taken ten years to shake out despite being put in with WG21's help. This was always intended to be a compatibility tool. We are repeating the same process that created the problem, not building on practice, especially without Microsoft. Make the changes in a TS or a study group.

Uecker: I disagree on experience; we have it, it is bad. I don't think any part of this needs additional experience because everything except the renaming exists. All the issues listed here have been pointed out by users before. The proposal can wait, but we urgently need eyes on it before adoption makes the problem too late to solve.

Sebor: so WG21 called for a library-only implementation, which existed and worked. The language-level features were added by WG14 without any prior art.

Ballman: I share Sebor's concerns w.r.t compatibility and implementation; Boehm had a paper, C++ finally made the qualifier work and we aren't coordinating with that; I do not want this in C23, I want to bring it to the SG.

Uecker: so right now the spec is fully compatible with C++, only the qualifier would change. The C++ side has `atomic_ref`, this is introducing a C counterpart to what C++ added without coordination.

Ballman: OK that means header compatibility is not so bad.

Gustedt: are you sure about the alignment issue? Saying *any* type can be used regardless of layout.

Uecker: there is already support for oversized types. This also works for misaligned types. You can use the specifier or `alignas` to get the right lock-free type. This is already an issue for `complex float`.

Gustedt: because we can't guarantee the alignment of a normal `double` array, we would always have to use the non-performant path anyway. Locally I align a bit-matrix on a small boundary.

Uecker: by definition in an existing data structure you cannot control this.

Voutilainen: on compatibility, C++ vendors attempted it. C++ wanted to inherit the C struct types, but that didn't work because it wasn't guaranteed to exist. We need more than a SG, we need cross-language experience to field-test the compatibility.

Uecker: incompatibility even exists between libraries on the same platform – there is already no ABI. The specifier still intends C++ compatibility as far as possible, nothing forces an incompatibility, but allows for atomic access to non-atomic objects. Users expect this feature and want this feature.

Voutilainen: users have been burned before.

Gilding: is it a problem if the new qualifier is sub-optimal? It doesn't break the specifier. This will often compile correctly already – it's most compilable, on the least-affected platforms.

Uecker: yes.

Banham: it implies asymmetry – pointers without the qualifier may exist in concurrent usage, which the existing specifier would allow safely.

Uecker: imagine within a procedure, using optimized access in a threaded way to a shared non-local object.

Banham: putting this on the interface to a function that internally uses concurrency is confusing. It places an invalid constraint on usage. There must be a better way of expressing this to the compiler – are you expecting the compiler to insert GPU operations?

Bachmann: not a fan of optional features, but since they already are – can we add another option to say that there are never embedded locks and ensure this is always alignment-based?

Uecker: the specifier can still change size, even if it is not lockfree.

Straw poll: (opinion) Does WG14 want something along the lines of Alternative A in n2909 in C23?

Result: 4-7-7 (no consensus)

Straw poll: (opinion) Does WG14 want something along the lines of Alternative C in n2909 in C23?

Result: 5-7-6 (no consensus)

Bhakta: what about not for C23?

Sebor: I would like the issues fixed without repeating our mistakes. I want collaboration with WG21 and implementations to get the compatibility right, not leaving it unaddressed.

Banham: I would like parallelism in-language going forward; dialects exist which are compatible with C++, but may diverge. There are a lot of SIMD extensions to consider.

Keaton: this is important enough to warrant a TS. It doesn't have to be big.

5.22 Ojeda, memset_explicit (updates n2682) [N 2897]

Ojeda: this is a revision of the previous version that removes unwanted parts of the proposal. Addresses concerns from Myers and Meneide about semantics outside the abstract machine; uses Gustedt's wording for implementation-defined internal copies. Alternative 1 is minimal; Alternatives 1b and 4 add more examples of implementation-defined options, with 4 adding the implementation-defined wording. "Unobservable" comes from Tong.

Bachmann: I would have proposed Alternative 1 myself, but am happy with any variant.

Bhakta: I don't like 4 discussing effects outside the abstract machine in the description. This can be in a footnote, but not normative text. OK with Alternatives 1 and 1b.

Wiedijk: I share Bhakta's concern. "by the abstract machine" should be "in". Would prefer 4b, with the abstract machine in a footnote.

Sebor: I agree but would go further – I only like Alternative 1, don't want the additional details of the other options. I don't think "the purpose" sentence is correct, the object is already inaccessible and destroying the data is the goal. Not an objection.

Preference poll: (voting for all acceptable options) Does WG14 prefer Alternative 1, Alternative 1b, Alternative 4, or none of the above?

Result: 13-10-8-3 (Alternative 1)

Straw poll: (decision) Does WG14 want to adopt Alternative 1 from n2897 as-is into C23?

Result: 15-1-7 (adopted)

Chat: thanks to Ojeda.

7.1.1 How to schedule after C23 (continued)

Keaton: the answer to alternating focus or not depend on the time between editions. We have options for 3, 4, 5 or 6 year schedules.

Gilding: is it possible to match the C++ schedule?

Keaton: it doesn't seem to matter. Sutter and the others don't think it is necessary.

Gustedt: I really want a faster schedule to avoid backlogs. Instead of feature vs bugfix, we should make a plan at the start of a release for features based on the existing backlog.

Voutilainen: the reason C's schedule doesn't matter is that C++ has a split within the cycle and feature-freezes early. This would freeze C early, so it favours a C Standard that is already released. C++ wouldn't base on a C Standard released late in its own cycle. It is worth considering this model; it gives time to force a focus after deciding which features we want.

Wiedijk: doesn't this depend on the Charter discussion? Make time for feature planning if we want them?

Keaton: features will happen either way, so, no.

Myers: I am concerned that saying up-front that we want features may encourage admitting features before dependencies are resolved; e.g. allowing lambdas before their usage is established. Allowing more time is OK, but lowering standards to get features in is not.

Keaton: wanting a feature is not a commitment to put it in.

Seacord: one thing we wanted fro C23 was to fix uninitialized reads, and we didn't put that in. A three-year cycle fits the modern world's CI/CD pipeline model; we should try to keep all cycles the same and try to settle into a long-term maintenance mode.

Uecker: I also favour a short cycle, prevents rushing to avoid long delays. Agree with Voutilainen about dedicated bugfix time within a cycle. Bugfixes matter, we don't want rushed features, or stalled development.

Sebor: we have no experience with short release cycles, this is a drastic cut to the length. Trying to be conservative with feature iteration, would want four years for `atomic_var_init`; not comfortable with anything less than five years.

Friday

5.23 Urban, C Identifier Security using Unicode Standard Annex 39 v2 [N 2932]

Urban: remaining issues in C and also C++, there is a very small risk of homoglyph attacks. This impacts readers and reviewers, not tools, but there is not much linting yet. Propose adopting the existing Unicode Annex 39. Conforming to only TR31 only ensures identifiers are well-formed, not that they are clear. This is a stable proposal for many years. GCC, Python and Go all have such modules: “confusable” checking also exists but is extremely slow, lots of feedback from users about the problems. Clang found tons of false positives in confusables.

This proposes to allow Greek but no intermingling with Cyrillic, disallows “secondary languages”. Specific range numbers change in every version, Arabic median-position removed in the latest. Some composable sequences can bypass the NFC rule. The biggest issue is political, with disallowing certain scripts, therefore want a pragma to enable scripts by name. Normal range is “good enough”. Implementation seems fast, Rust has it. C remains low-risk as dynamic languages are most vulnerable to injections. Only 10% of the range is now allowed. Need to split characters by script now, but the table isn’t too big. What is a “common script” is a political issue. Arabic and Hebrew are RTL; CJK characters can coexist with Roman; Roman may mix with Greek but not Cyrillic. The list for Greek is partial. GCC is now shipping this; MSVC and Clang have implementations that are off-by-default.

Krause: in C we value backwards-compatibility. We need to decide for C23 and set a precedent before users start writing code that this would ban. The pragma should be standard, not implementation-defined – that is too obvious to leave out. This is a huge implementer burden. The reference implementation is APL2 licence, which is not compatible with GPL2 and not permissive enough for BSD.

Keaton: we cannot put it in C23, this came in too late, it is a scheduling error.

Urban: we would be happy to relicense the reference implementation. Agree that the pragma should be standardized.

Myers: we don’t have text changes and are out of time for C23. C++ are too eager to retroactively apply defects – they should be simple, tiny, and a few lines, not huge new normative references. C99 made choices; C11 also made choices that were reasonable in context; for C23, when one reasonable choice becomes another that is *not* a defect report. I do not think a programming language standard is the place for this – it is a text issue, not a C or C++ issue, it could also have logical issues with key-strings; this should be added to text editors and review tools. Some confusables wouldn’t be covered at all: long identifiers differing by just one letter are at least as hard for a human reviewer to spot. Other tools like spellcheckers do this better.

Keaton: in ISO there are no defect reports against withdrawn versions.

Honermann: this doesn’t adhere to current Unicode guidance, saying Unicode is wrong without justification. Anything here is premature, Unicode has a new source code ad-hoc study group working on upgrading Unicode to target all users. There was discussion of Swift for identifier restrictions in P1949: it should work on the whole ecosystem. The Unicode WG agrees that Annex

31 is good for programming language standards but that 39 is not, and should change over time to be better for tooling. This might be workable for C26.

Urban: the claimed languages are in the appendix; we filed bug reports against TR31, indistinguishable half-width letters can't be identifiers. There are operators in `ID_Start`, two ranges in Arabic that look like the base but are only for printing, not for identifiers. I agree with Myers that long names are a problem, especially in C with generative code; these are difficult to shorten usefully. w.r.t bidirectional and stateful characters, these can bypass all syntax including quotes and comments, the origin of Trojan Source. Identifiers are not such a risk. We should consider filenames, but the Linux kernel doesn't care; Apple had this but removed it.

Gilding: I disagree that kanji are not in common use, we have many customers who write C with heavy use of CJK characters in identifiers. The implementation for these checks isn't too heavyweight, but it is a non-zero burden. In a MISRA context for Directive 4.5, implementation as-written is prohibitively non-performant and this is a better replacement for that rule.

Bhakta: agree with Myers and Gilding. I like the pragma and would put it in the Standard. But lots of customers want multiple scripts in one source file, we cannot say this allow-list would be enough.

Urban: the Indic scripts do have a high mix, up to five at once is common.

Bhakta: our customers wanting multiple scripts are mostly European…

Urban: clearly more customer feedback is necessary.

Ballman: what is the teachability of this? The current rules are teachable, this is not. We have already started getting user comments on TR31, and will get a lot more. Users of mathematical script will not like this. We need a way to gauge how many users will break.

Urban: we have been using this in Perl for about seven years and math symbols are fine. Mixing with Cyrillic is especially a problem. We have feedback from users using Greek.

Straw poll: (opinion) Does WG14 want something along the lines of N2932 in a future C version?

Result: 2-10-6 (no consensus)

Urban: I do have a better third version to show to C++…

Honermann: target the Standard vs targeting tools?

Keaton: maybe this is good material for a TS.

Straw poll: (opinion) Does WG14 want something along the lines of N2932 as a TS?

Result: 9-1-7 (clear direction)

7.1.1 How to schedule after C23 (continued)

Keaton: we don't have to resolve this now. Take the time to think about it until the next meeting. Mixed vs. targeted cycles; 3, 4, 5 or 6 years. We are close to resolving that we do not want any "hard" themes, only a soft focus, if at all. We have the option for separate deadlines.

Bhakta: I believe long cycles are better, like Sebor. Bugfixes are more important than features – C is chosen for its stability. We would lose the differentiating feature of C. Failed features have always been inventive.

Myers: I tend to support the soft-alternation system – it is better to iron out bugs. We need a way to queue and bring back papers for features proposed at the wrong time. Previously we have "lost" features proposed at times we didn't have a WD open – we need tracking to fairly consider features.

Wiedijk: I echo Bhakta's position, the Standard is for what is already there. There are many different kinds of proposal, Voutilainen proposed things more C-like than other proposals.

Sebor: what is the origin of the 3-6 year constraint? ISO?

Keaton: because we wanted it to be fixed – ISO has a rule that a WG will be disbanded without a project in progress. We cannot assume another document will keep the group alive.

Sebor: I am concerned that even a 5-6 year schedule is too aggressive. Is there a way to make it happen?

Keaton: the Standard itself takes three years once we say we are revising it. A Preliminary Work Item can gain another three years for a six year maximum.

Meneide: a lot of practice came up in past discussions in 2014-2019. Features like Statement Expressions couldn't be added without a WD in progress, therefore a hard limit is a bad idea – we lose interested experts along the way. Soft focuses can prevent a feature getting lost because the person leaves during the long wait.

Myers: w.r.t Sebor wanting long cycles – the relevant cycle is the *sum* of the bugfix and feature cycles, rather than either individually.

Bhakta: we don't follow our own rules anyway so limits aren't an issue, we even violated them in C17, I don't expect the distinction to be meaningful and it's better to move this along.

Sebor: agree on following our own rules. If we change, Myers's idea is appealing. We should put more work into TS and less direct adoption into the Standard, helpful to gain experience before adoption. This extends the time to unlimited on a feature basis.

Keaton: we have more options – affirm, withdraw, promote to IS, or merge to the Standard.

5.2 Discuss how we wish to apply the C23 Charter [N 2611] (new additional paper by Sebor [N 2986])

Sebor: what is the problem? We have an outline of Principles, but the proposals are evaluated while at odds with it. The Charter is a high-level outline of goals and principles the WG followed and intended to follow, updated for each revision cycle. The goal was to establish acceptance criteria to evaluate proposed changes fairly, with the principles as the baseline criteria for changes.

Two principles are core: C99 added principle 8, "codify practice", for concepts with prior art – the last sentence, "no invention", leaves room for novel features that address deficiencies. C11 added principle 13, because C99 was lacklustre: no invention at all, eliminates novel features to address deficiencies. Requires a history of customer use. C23 adopted essentially the same. Past adherence: C99 had the most latitude but was disciplined, every new feature came from implementations. C11

added principle 13 then added new features, many problematic. Threads and Annex K were not implemented.

For C23, we restated but have moved in the opposite direction, proposals lacking prior art being put into C. Proposals themselves are powerful and excellent but don't have implementation or user experience, large amounts of time spent refining features without any shipping work or even diverging from it. We survey proposal papers to illustrate adherence; if the Charter is to mean anything this must be reconciled. Amend, affirm, and then adhere with active oversight by the convenor to focus on the principles and require deviations to be addressed.

If there is insufficient support we should amend or remove principles 8 and 13 to reflect our position. We need to continually remind ourselves, maintain consensus, and enforce provision and documentation of rationales for deviation; we need a volunteer to monitor and amend the submission guidelines, require adherence to the format and the rationale.

Gustedt: this misses the point of diverging implementations. w.r.t lambdas, we have explained diverging practice with multiple, irreconcilable implementations like nested functions; WG14 settled on lambdas as the route to go down. We need to unify and address divergence, and resolve improvements.

Gilding: I posted a list of concrete proposals to modify the Charter on the reflector, which I will not repeat now; mostly, we need to add two SGs to act as a screening group and wording finalization group so that Committee time is not spent on wordsmithing once we have technical consensus, and that proposals that make it to the Committee are treated fairly. The role of the Standard vs. implementations has changed and implementations will no longer change or add features first without guidance in some form from the Committee; the time of implementation-led feature development is mostly over.

Bhakta: the Charter is irrelevant without enforcement. Even straw polls are overturned, sometimes immediately. I disagree that implementations need guidance; Clang accepts lots of new features, implementations still do new things. If a paying customer asks IBM for something, we *will* do it.

Myers: applying a modified version of tested implementation experience depends on what is relevant to the proposal. For some features, different methods or experience in other languages may be enough, especially to explore the design space. For others test cases and implementation in a C context is much more important; depends on the level of interaction with other features. It is not yes/no, but need to balance the principles and when the experience is important.

Ballman: agree with Gilding w.r.t extensions: Clang documents and enforces *no* invention. "Not your playground" – extensions need to go through an authority. Extensions do get added, attributes are easy, while lambdas would be implausible. It is an iterative chicken-and-egg process.

Bhakta: extensions do get in; WG14 can only control one authority. Clang is widely forked and extended, Clang's guidance is violated as much as the Charter is. C23 already has at least two features breaking existing code. Consider other proposals as well – do we care about existing code?

Sebor: I acknowledge the chicken-egg issue – GCC is similar, especially if it touches the language grammar. Small enhancements get in and experimental mechanisms exist, similar to the mechanisms in ISO to support experimentation with TS, giving the opportunity to implement. We need experience to understand the interactions that are there, so going the TS route is worthwhile. There may be exceptions but we need to follow the principles or they are meaningless.

Meneide: echoing Ballman, as a non-implementer, I worked on `#embed` for years and even the underlying `__builtin`s would not be accepted by GCC. I have maintained patches supporting commercial teams, which did not prove adequate. The only way to get practice will be to build a full commercial implementation, which I *can* do, but effectively rules out normal user contributions to C altogether. If so, so be it, but this needs to be made clear.

Ballman: I support this point – the Charter and Committee don't make clear what is an implementation. Forking a compiler is "implementing" it, but without users it doesn't tell us about practical uses. There is no recourse to bring ideas to the core maintainers who want signoff first. Is the Committee even serving implementation needs here? A SG to screen good ideas so that implementers get early direction might help.

Bhakta: I agree with Meneide, but no commercial implementer will say no to money. DO we want to standardise practice? If so, a hard battle. If not we have more leeway.

Sebor: the Charter sets a deliberate high bar for compiler proposals; it may be worth relaxing for library features which can ship separately. For compiler features, if a user submission is turned down, that rejection *is* the intended bar. It does pose a problem w.r.t inventive enhancements; we need to decide and then enforce.

Uecker: I largely agree with Sebor, but the bar seems far too high; I have tried to get features into GCC before and it is a very hard bar for users. The Standard is a group of multiple stakeholders and should not just close the door. Ideally we should move large features into a TS first and drive implementations this way. Principle 13 is too strong – the rest of the Charter mentions balance, 13 is exclusive.

Keaton: ironically the pandemic helped here. When the C2x Charter was written in 2016, only two NBs reliably attended, plus guests; only four attended in Ithaca. Five are needed for a NWIP! TS used to be very difficult to start, had to drive hard for one. Now the virtual meeting has 11 Nbs, 10 present for TS creation this week, easy to poll. The "new" mechanism is much easier to use for features, giving us the four options to progress a TS. The Charter was written for the old world.

Bhakta: that's great, didn't know the history. I disagree with the chat channel that this only works for Library – Decimals added a huge language part; though I agree that it's easier for Library.

Ballman: clarify that agreeing for Library was specific to WG21 experience – it is large in both, but the language TS didn't gain adoption. Continuing, the problems may not repeat. The language TS has much higher expense and early adopters balance expense against the possibility of WG14 changing it anyway, which is not so bad for library, we have linkers etc. So it's not a silver bullet until we know what implementers are willing to do. TS 6010 is an invasive change that will be indicative.

Myers: on language vs. library, library features don't have to have critical experience, also it's easier to gain experience and can be useful standalone.

Sebor: much C++ experience with TS; I wish to hear about core language TS successes. Experience is that implementers are willing to work with proposers in C++. There is incredible value even in a fork just to get user exposure, subject to shipping. If a TS doesn't work in C++, what does?

Bhakta: they add stuff and let it break. From the WG14 side, we have had success with language TS, in terms of seeing what community does and integrating if they implement it – Decimal, Embedded C, etc.

Voutilainen: I don't think language TS are useful – we had transactional memory, concepts, modules – no implementation of modules, but competitors instead. Only one implementation of concepts. The scale is different – *so* much stuff that C++ vendors are swamped. It's not been useful for gaining deployment experience. C++ doesn't have the strict requirement and doesn't get experience before shipping.

Ballman: C++ had five language and twelve library TS; nobody implemented the modules TS, with concepts the implementation pre-dated the TS (strange experience). Coroutines actually did get experience and three implementations but are much smaller. There are two transactional memory TS and we haven't seen either implemented, though maybe these are staying to the side like Embedded C.

Keaton: a TS is an experiment, all answers are successes. No implementation is still an answer providing information. I would like a review of the Charter with explicit mention of TS as part of the strategy; this affects the balance of principles.

Bhakta: I like that, but want to see meaningful enforcement. Is that a concern?

Keaton: balance was different when TS were expensive.

Bhakta: so we will follow the Charter better with TS in there.

Sebor: so for C2y, or C23? I intended this discussion to be for C23. We need to reconcile the outstanding proposals.

Keaton: it is too late to do for C23. Everything still on the list has had an indicative vote, we cannot change that.

Bhakta: it isn't fair to change the rules midway through the release.

Uecker: is there a risk of a return to the pre-pandemic status quo? Probably why we have so many proposals is that they are reflecting a need from the community. Vendors can say no unless money is spent, but that's not entirely true: the existence of a proposal indicates community need for something. It may be better to collaborate between the WG and the open source community to help things get through – user experience getting things into GCC was hard, but that doesn't diminish the use cases.

Voutilainen: it's a good idea to try the TS approach. We have different historic experience in C++, with the variable standardization cycle – a three-year cycle avoids aggressive adoption drives, and doesn't take too long to reopen ideas. A TS may also open parallel paths and an option to detach mid-cycle.

Bhakta: anyone can propose anything to WG14; in general that only indicates that one author cared, not a community need.

6. Recommendations and Decisions reached (Agenda 8)

6.1 Review of action items

ACTION: Marcus Johnson to add a request to place the final version of the Unicode Length Modifiers proposal on the Papers of Interest list.

ACTION: Aaron Ballman to put N2829 "friendly assert" on the Papers of Interest list.

ACTION: David Keaton to request an extension to the C23 schedule at SG22 plenary.

ACTION: JeanHeyd Meneide to make editorial change from "floating multiply-add" to "fused multiply-add".

ACTION: David Keaton to submit a NWIP to SC22 for "C - Extensions to support generalized function calls" (proposed as N2976).

ACTION: Aaron Ballman to write up the issue tracking process next-steps and post to the Reflector.

ACTION: David Keaton to start the DTS ballot process for TS 6010.

ACTION: David Keaton to notify Peter Sewell that we are delaying the ballot to make the change in N2889 to TS 6010. (**DONE**)

6.2 Review of decisions reached

Decision: Do five NBs commit one person each to participate in "C - Extensions to support generalized function calls" (proposed as N2976)?

Yes (Austria: Bachmann, Canada: Bhakta, France: Gustedt, Germany: Uecker, UK: Myers, US: Gilding/Bhakta)

Decision: Does WG14 want to adopt N2867 as-is into C23?

19 / 1 / 2 (adopted)

Decision: Does WG14 want to move to a DTS ballot for TS 6010?

18 / 0 / 0 (adopted)

Decision: Does WG14 want to integrate Variant 1 of the change specified in N2886 into C23?

14 / 2 / 2 (adopted)

Decision: Does WG14 want to exchange Variant 1 with Variant 2 of the change specified in N2886 in C23?

9 / 3 / 6 (adopted)

Decision: Does WG14 want to integrate Change 3.5 in N2886 into C23?

8 / 1 / 9 (adopted)

Decision: Does WG14 want to integrate the Recommended Practice in Change 3.6 of N2886 into C23?

2 / 5 / 9 (no consensus)

Decision: Does WG14 want to integrate changes 3.2, 3.3 and 3.4 from N2888 into C23?

12 / 1 / 4 (adopted)

Decision: Does WG14 want to integrate changes 3.1, 3.2 and 3.3 from N2889 into C23?

4 / 7 / 7 (no consensus)

Decision: Does WG14 want to delay the DTS ballot for TS 6010 and integrate changes 3.1, 3.2, 3.3 from N2998 into TS 6010 before balloting it?

8 / 3 / 7 (adopted)

Decision: Does WG14 want to adopt N2861 as-is into C23?

13 / 1 / 3 (adopted)

Decision: Does WG14 want to adopt N2992 as-is into C23?

16 / 0 / 3 (adopted)

Decision: Does WG14 want to adopt Alternative 1 from N2897 as-is into C23?

15 / 1 / 7 (adopted)

7. PL22.C Business (Agenda 9, previously PL22.11)

7.1 Identification of PL22.11 Voting Members

7.1.1 Members Attaining initial Voting Rights at this Meeting

None.

7.1.2 Members who regained voting rights

None.

7.2 PL22.11 Voting Members in Jeopardy

7.2.1 Members in jeopardy due to failure to vote on Letter Ballots

Cisco, in jeopardy since 2010.

7.2.2 Members in jeopardy due to failure to attend Meetings

7.2.2.1 Members in jeopardy who retained voting rights by attending this meeting

None.

7.2.2.2 Members in jeopardy who lost voting rights for failure to attend this meeting

None.

7.3 PL22.11 Non-voting Members

7.3.1 Prospective PL22.11 Members Attending their First Meeting

None.

7.3.2 Advisory members who are attending this meeting

None.

7.4 Other Business

Tydeman: what about approval of the minutes? Maybe it would be better to group all C business together?

Hedquist: I prefer putting this at the top.

Keaton: this used to be in Agenda 9 but we already requested the move to the top.

8. Thanks to host

Thanks to ISO for providing Zoom capabilities.

9. Adjournment (PL22.C motion)

Tydeman moves, Pygott seconds. No objections.

Adjourned.