Draft Minutes for 27 and 30 – 31 August, 1 – 3 September, 2021

MEETING OF ISO/IEC JTC 1/SC 22/WG 14 AND INCITS PL22.11

WG 14 / N 2803

Dates and Times

Each day will have a half-hour break from 15:00-15:30 UTC.

Fri 27 Aug, 2021	13:30 – 17:00 UTC
Mon 30 Aug, 2021	13:30 – 17:00 UTC
Tue 31 Aug, 2021	13:30 – 17:00 UTC
Wed 1 Sep, 2021	13:30 – 17:00 UTC
Thu 2 Sep, 2021	13:30 – 17:00 UTC
Fri 3 Sep, 2021	13:30 – 17:00 UTC

Meeting Location

Please note: Due to the global health emergency, this is no longer a face-to-face meeting.
This meeting is virtual via Zoom.

Meeting information

Please see the ISO Meetings platform (log into login.iso.org and click on Meetings) or contact the convener for the URL and password.

Local contact information

David Keaton <dmk@dmk.com>

1. Opening Activities

1.1 Opening Comments (Keaton)

Svoboda will take minutes.

1.2 Introduction of Participants/Roll Call

Name	Organization	NB	Notes
Bill Ash	SC 22		SC 22 manager
Aaron Bachmann	Austrian Standards	Austria	Austria NB
Roberto Bagnara	BUGSENG	Italy	Italy NB, MISRA Liaison
Aaron Ballman	Intel	USA	C++ Compatibility SG Chair
Dave Banham	BlackBerry QNX	UK	MISRA Liaison
Rajan Bhakta	IBM	USA, Canada	PL22.11 Chair
Lars Gullik Bjønnes	Cisco Systems	USA	
Alex Gilding	Perforce / Programming Research Ltd.	USA	
David Goldblatt	Facebook	USA	

Name	Organization	NB	Notes
Jens Gustedt	INRIA	France	France NB
Barry Hedquist	Perennial	USA	PL22.11 IR
David Keaton	Keaton Consulting	USA	Convener
Philipp Krause	Albert-Ludwigs-Universität Freiburg	Germany	
Paul McKenney	Facebook	USA	
Kayvan Memarian	University of Cambridge	UK	
JeanHeyd Meneide	NEN	Netherlands	Netherlands NB
Maged Michael	Facebook	USA	
Joseph Myers	CodeSourcery / Siemens	UK	
Miguel Ojeda	UNE	Spain	Spain NB
Clive Pygott	LDRA Inc.	USA	WG23 Liaison
Mark Santaniello	Facebook	USA	
Robert Seacord	NCC Group	USA	
Martin Sebor	IBM	USA	
Peter Sewell	University of Cambridge	UK	Memory Model SG Chair
Eskil Steenberg	Quel Solaar	Sweden	Sweden NB
David Svoboda	CERT/SEI/CMU	USA	Scribe, UBSG Chair
Fred Tydeman	Tydeman Consulting	USA	PL22.11 Vice Chair
Martin Uecker	University of Goettingen	Germany	
David Vitek	Grammatech	USA	
Ville Voutilainen	The Qt Company	Finland	Finland NB
Freek Wiedijk	Plum Hall	USA	
Michael Wong	Codeplay	Canada, UK	WG21 Liaison
Victor Yodaiken	E27182	USA	
Hans Boehm	Google		Guest
Steve Downey	Bloomberg		Guest
Corentin Jabot	Freelance	France	Guest

1.3 Procedures for this Meeting (Keaton)

Keaton: I have made Bhakta a co-host in case I get knocked off the Internet. I have also pasted my cell phone number in the chat session. If you get knocked off when you want to discuss something, please call me.

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 2768] (WG 14 motion)

Svoboda: I have applied Tydeman's errata. I will submit a new revision after this meeting. Also, Tydeman says: Should section "9 PL22.11 Business" be listed in the WG14 meeting minutes N2768? It is removed from the previous minutes.
Bhakta: Add the heading & nothing else so it does not look odd.
Svoboda: I will add just the heading.
Ballman: I move to approve the minutes, modulo adding a heading for section 9
Gilding: Seconded
Keaton: Any objections? (none)

1.6 Review of Action Items and Resolutions

- *Keaton*: will investigate problems with the WG14 reflector. (done during meeting)

  Done

- *Keaton*: Put standardization schedule as an agenda item on the full January agenda.

  Done

- *Tydeman*: Respond to Austin Group about whether the remquo result should be a NaN.

  Ask later when Tydeman is here

- *Bhakta* & *Ballman*: Have CFP SG target a paper to WG21 summarizing changes to C floating-point.

  Bhakta: Done.
  Ballman: Yes, will be discussed in our next meeting in 2 weeks.

- *Keaton*: Publish a document clarifying meeting schedule. The next two meetings are both virtual. First meeting is Friday August 27 plus the week of Monday August 30, with the second meeting happening the week of November 15.

  Keaton: Done. We will examine shortly. It is document N2759.

- *Svoboda*: Submit a document that makes the "ckd_" identifiers from the (now accepted) Core proposal in N2683 in future library directions to be potentially reserved identifiers.

  Svoboda: Done.

- *Keaton*: See what actions we need to take with ISO in order to start the TS 18661-5 revision. (This is done.)

  Keaton: Done

- *Keaton*: Get SC22's permission to revise TS18661-5.

  Keaton: Done; it will be discussed shortly

1.7 Approval of Agenda [N 2791] (PL22.11 motion, WG 14 motion)

Ballman: I move that we approve the agenda.
Pygott: Seconded
Keaton: Any objections? (none)
Bhakta: I have a short list of potential agenda items to discuss if we have time.
Keaton: I generally order them starting with most-likely to discuss. However, this ordering is disturbed by people being present.

1.8 Identify National Bodies Sending Experts

Austria, Canada, Finland, France, Germany, Italy, Netherlands, Spain, Sweden, UK, USA

1.9 INCITS Antitrust Guidelines and Patent Policy

1.10 INCITS official designated member/alternate information

Ballman: Melanie Blower has retired from Intel, we will bring someone else in shortly.

1.11 Note where we are in the C23 schedule [N 2759]

Keaton: This is a 6-day meeting, but the November one is currently 5 days, although we will vote this week on whether to extend it to 6 days.
Ballman: If a paper comes in after the mailing deadline but before the meeting, will that be considered?
Keaton: I still want the January meeting to be non-virtual. I am working with Herb Sutter (of WG21) to coordinate this.
Gustedt: Can we have more meetings outside the US, as travel there is difficult right now?
Keaton: Noted. Please email or let me know your other concerns with regard to the January meeting (which is currently face-to-face in Portland).
Ballman: Has there been process slippage on ISO's side for the DIS?
Keaton: This was a problem, but they are getting better, and they will not cancel our project if this turns out to be problematic.
Bhakta: Is there a deadline for final proposals of new features for C23?
Keaton: The deadline for documents for the Portland (January) meeting will serve as that deadline.
Ballman: I hope that we can bring back the defect report / clarification request process, rather than writing a full paper.
Keaton: We do need to have a discussion on what that process will be. Our current job is to get through the new proposals.

2. Reports on Liaison Activities

2.1 ISO, IEC, JTC 1, SC 22

Keaton: The Floating-Point TS parts 4 & 5 were approved for revision, so CFP can work on those. Also, the C part of the WG23 Vulnerabilities TR was also approved for revision. I expect we will work with WG23 on that.

2.2 PL22.11/WG 14

2.3 PL22.16/WG 21

Ballman: WG21 had a plenary virtual meeting in June. Highlights: New work item for version 2 of Concurrency TS. We are focused on hazard pointers. With regard to WG14, there is a significant amount of effort to clean up Unicode support. WG21 has SG16 which focuses on cleaning up Unicode support. I am encouraging proposal authors to present them to WG14. There are many requests for help for how to follow WG14 procedures. Can we ease the process for WG21 members somehow?

2.4 PL22

Keaton: We are currently considering new work item proposals for WG23 documents.

2.5 WG 23

Pygott: We are currently promoting existing documents from TR's to IS's, so we can apply to have them made freely available.
Keaton: This is being done because ISO no longer allows TR's to be freely available. Also ISO no longer allows TR's to contain guidance, which is what these documents do.
Gustedt: Does this mean TR 6010 will not be freely available?
Keaton: 6010 is a TS not a TR. It will not be freely available, but it is not the kind of thing ISO would make freely available anyway.

2.6 MISRA C

Banham: MISRA is updating for C11. Then we turn our attention to C17.

2.7 Austin Group

(Stoughton is not here)

2.8 Other Liaison Activities

3. Study Groups

3.1 C Floating Point activity report

Bhakta: We are looking at corrections to the C standard & TS's. Part 4b and part 5 are based off C23 if we can get it done in time, otherwise we will just support C17. Also, we will present a 1-page summary to C/C++ SG; we would welcome feedback there. We would

like some C++ expertise in the SG.
Ballman: There is a WG21 Study Group: SG6, related to numerics. I've mailed Bhakta emails for their chairs. They slowed down work because of COVID19, but are picking up again.

3.2 C Memory Object Model Study Group activity report

Bachmann: There is much discussion about initialized values on the reflector. There will be three proposals on Tuesday based on that discussion.

3.3 C and C++ Compatibility Study Group activity report

Ballman: Omnibus of WG21 Papers (Jun 2021) [N 2753]. We have once-a-month meetings. 5 papers processed so far. We are currently focusing on character sets. One early feature arises, about making arrays be first-class object types, so you can use assignment on them. This is WG21 proposal P1997. More members would be welcome.

3.4 Decide whether to create a Study Group for undefined behavior

Keaton: There might be support for creating an Undefined Behavior SG based on the reflector conversation.
Gustedt: What would be the goal of this SG?
Keaton: The SG would decide if we should create TS's for mitigating undefined behavior in safety- or security-critical environments. The SG could recommend whether to publish a TS. The SG could create a working draft of such a TS.
Ballman: Would there be overlap between us and the Memory Object Model SG?
Keaton: Yes, the charter should define boundaries between the two SG's.
Steenberg: We can start out by figuring out which UB's are a problem. They are not always related to safety or security.
Keaton: That would be a prerequisite, and addressed early on. Part of the "approach to a TS" part of the charter.
Wiedijk: This sounds very close to the Memory Object Model SG, and also quite technical, involving how the C community (mis)understands things. This charter does not, but could, address social aspects. Undefined behavior is a PR problem with people not understanding it well.
Keaton: That might be worth words in a TS. It is a good idea, but I would prefer to let the SG decide that.
Gustedt: Does this need to be separate from the Memory Object Model SG?
Keaton: This is about undefined behavior in general, not just memory.
Bhakta: I did not see it as a safety-security thing. A wider scope would help get members interested & keep momentum. Perhaps we should remove "safety/security" from the charter?
Seacord: Separation of undefined behavior into separate types make sense. Perhaps we should change the standard rather than publish a TS. A TS would not change anything; it would just clarify existing practice. We should subdivide undefined behaviors into categories, a la Annex L.
Keaton: For big changes such as the definition of undefined behavior, we tend to use a TS before merging into the standard
Svoboda: I would say that all undefined behavior is of interest to people interested in security (and probably safety too). I know of no undefined behavior that is not security-related. Yes, a SG would need better PR, and perhaps a rationale document. We cannot publish a TR if TR's cannot give guidelines/recommendations.
Keaton: Guidance can be in a TS or IS, just not a TR.
Goldblatt: Compiler optimization writers have less appetite for restricting undefined behavior. We do not want this group to be ignored by them.
Keaton: If you write compiler optimizations, would you please participate?
Pygott: Compiler writers rely on undefined behavior for their optimizations. We do not want to make code slower.
Keaton: Instead of deleting "safety/security", perhaps we should replace with "including environments concerned with safety/security"?
Ballman: WG21 has a group: SG12 which studies undefined behavior. They have been around approximately 6 years. Instead of trying to produce TR's or TS's, they audit undefined behaviors to see if they can modify or "fix" them. That might provide better member participation.
Keaton: That sounds like something to add to the charter. That is the job of an ongoing SG. This could be another purpose for this SG: to monitor ongoing changes to the standard. I will draft a revision to this charter that we can re-examine later in the meeting.
Myers: Optimizations are often done on the SSA level, so it is hard to relate SSA changes to C undefined behavior. Second, platforms can provide different implementation modes to mitigate undefined behavior. WG14 does not like defining multiple language dialects. Having source code specify how to handle undefined behavior might be better than creating language dialects. There are various undefined behavior compile-time/run-time detectors…perhaps we can standardize those?
Keaton: WG14 does not dislike different modes; we did create Annex L. The users did not demand Annex L. We need to keep working on that space until people demand that their compilers support things.
Bhakta: Implementations matter. Annex L was not a success, so let's not waste our time on what the user community does not want. Implementations should be a strong consideration for what we should consider. Also, please do not use Zoom chat as a replacement for the meeting, it is for reference. Pay attention to the speaker instead.
Bhakta: Discussion of undefined behavior often goes into holy wars, ignoring either safety concerns or market concerns. That is a hazard

the undefined behavior SG should avoid. So I would want to not be a member of this group.
Svoboda: We need to balance safety/security vs. compilers. That is, avoid holy wars. Let us take a bipartisan approach.
Ballman: The chat lets you post links. WG21 uses the chat as a side-channel, but it is not part of official discussion. It does get posted to minutes wholesale, and this topic comes up on every single meeting.
Svoboda: The chat can contain sensitive info like email addresses.
Keaton: Use it for info like URL's and augment the conversation. Do not use it to replace the conversation. I will also update the charter based on our discussion.

4. Future Meetings

4.1 Future Meeting Schedule

Please note that in-person meetings may be converted to virtual meetings due to coronavirus considerations.

15-19 November, 2021	Virtual, 14:30-18:00 UTC each day
31 January - 4 February, 2022	Portland, Oregon, US (tentative)
11-15 July, 2022	Strasbourg, France (tentative)

4.2 Future Mailing Deadlines

Note: Please request document numbers by one week before these dates.

Post-Virtual-202108	24 September 2021	
Pre-Virtual-202111	15 October 2021	(This is the last chance for new C23 proposals)
Post-Virtual-202111	10 December 2021	
Pre-Portland	31 December 2021	
Post-Portland	25 February 2022	
Pre-Strasbourg	10 June 2022	
Post-Strasbourg	5 August 2022	

5. Document Review

Friday, 27 August

- 5.1 Working draft updates (as available)

  Tabled, as Meneide is not here.

- 5.2 Ballman, __has_include for C [N 2673]

  Gilding: It is necessary to support nontrivial major compilers & standard libraries. If this is available for standard libraries, should it not also be available for nonstandard libraries?
  Krause: Does this handle cases where a header file supports only a subset of functions, or is merely a stub?
  Ballman: __has_include only tells you if the file exists on disk. It does not know if the file is a proper include file. So people can do crazy things like an empty header file.
  Wiedijk: Do all the compilers you cite use __has_include? Do any use other names?
  Ballman: Yes, all are using __has_include. I do not know if other names are ever used.
  Bhakta: I am in favor of this. It can be helpful for external libraries, but for standard libraries, you have macros to say whether the standard library includes things, so it is not critical. Also, these may be logical files, but not physical files, so I am not sure this wording is correct: "a search for the file succeeds / fails".
  Ballman: We could use "headers" if that is more suitable for C. The text was lifted from C++ verbatim.
  Myers: You will also need to update s6.4p4 about context: that was either not an issue in C++17, or was fixed in C++20.
  Bachmann: Must the file be readable, or merely existent?
  Ballman: Existent. Voutilainen, did that question come up in WG21?
  Voutilainen: I think we only proposed "existent". We have field experience: we have implementations that make header files empty

when not quite right.
Keaton: There are some places in the wording where "has include" are separate words, which is confusing.
Ballman: That is a style choice that WG14 made. When using it as grammar in prose, you are supposed to remove the hyphens. I do not mind if you want to change that.
Gilding: The wording "search fails" is consistent with an include file actually missing.
Gustedt: Talking about "source file" is not right here. Removing "for the source file" would be much cleaner.
Myers: We already have "static_assert" where the underscore is preserved in prose. We could do __has_include to bypass the hyphen problem.
Banham: Could the example be less trite?
Bhakta: With regard to Jens' comment: I agree with the idea, but the existing text for #include says "source file". If we want to change that for __has_include, we should change it in both places or no change at all.
Myers: With regard to the example in C++, you need to adjust the header names to be C style.
Wiedijk: Does this poll affect what happens if we do not get wording in time?
Keaton: Ballman takes this as homework, and publishes it somewhere. It should then be a five-minute vote.
Ballman: If this does not make it into C23, yes we are inconsistent, but we have been inconsistent since C++17. Perhaps we can re-address this in Other Business later in this meeting.
Opinion Poll: Does the committee want something along the lines of N2673 in C23? 19-0-0 Clear sentiment to add this to C23

- 5.3 C++ liaison: Jabot, Delimited escape sequences [N 2785]

Bhakta: Curly braces are one of the non-invariant characters. Strings can be read in as distinct code-pages. It would be nice if we could use parentheses instead of braces. We can live with curly braces. We like this, but want to see implementation experience, and accept it afterwards. Not because this feature is bad; I just want to standardize existing implementations rather than invent.
Ballman: I would like to see a poll that ties down the syntactic portion of it. That would clear the path forward for this paper for WG21. If it passes, we can talk about an "along-the-lines-of-C23" poll. But I am OK if C++23 gets this and C23 does not.
Myers: In C we should include in constraints that the number does not exceed 0x10FFFF or it is a constraint violation, to make things consistent with C++.
Ballman: Thanks, I forgot to mention that.
Krause: Using curly braces inside string literals can interfere with digraphs.
Bhakta: That was my point with string literals.
Gilding: If you could not take curly braces outside string literals, you could define begin/end a la Pascal.
Jabot: Implementations can decide to use digraphs.
Bhakta: Yes, that is correct for our implementation. But in the C++ Standard trigraphs were removed, but not in C.
Ballman: Trigraphs are still supported by everyone in C++.
Banham: We could use [] or () instead of {}; we just need to agree with WG21.
Jabot: We could use something else, but would be inconsistent with other languages (even if we adjusted C++).
Ballman: In WG21 there are other proposals that backslash character things with things like Unicode names. The feeling is that they expect curly braces. Changing syntax would be disruptive in WG21. The question should be "Can WG14 live with curly braces?".
Opinion Poll: Would WG14 be willing to accept using curly braces to delimit escape sequences as described in N2785? 17-0-5 Clear direction
Gustedt: Would you have deployment before Portland?
Ballman: Deployment was waiting for WG14. We can have it before Portland if the poll is favorable.
Opinion Poll: Would WG14 want to adopt something along the lines of N2785 to be adopted into C23? 16-2-3 Clear direction

- 5.4 Santiago, Standard library should have a fuzzy way of comparing memory blocks [N 2684]

Krause: This is a way to do fuzzy comparison. I do not like the name. It should reflect the specific data format. Also the proposal should not use void pointers, and it should have some knowledge of the memory layout.
Myers: I agree. This seems specific to one particular form of comparison (bitmaps). There are many different forms of comparison for different purposes. This is not a general-purpose comparison.
Gilding: What is the goal of the standard in adopting specific things? A function like this demands a direction and that direction should be explicit.
Voutilainen: This feels like "please solve my problem and have this solution be available everywhere". It belongs in a library but feels like a poor fit for the standard library.
Gustedt: Is there prior art in a standard library?
Keaton: I was unable to find prior art in the C library, but there is prior art in many applications.
Svoboda: General-purpose or specialty applications?
Keaton: General-purpose, but only some of the implementations would be addressed by this proposal and others would not be.
Gustedt: There is no normative wording, and no terminology.
Keaton: Yes. For now it might be suitable for an along-the-lines-of poll. This proposal is for something that has to be implemented in many applications, but it is still a specific problem. an along-the-lines poll might be appropriate.

Svoboda: I disagree. I would want a paper that provides a more generalize-able solution, and that desire is not captured by an along-the-lines-of poll.
Bagnara: We need a rationale for including this in the standard.
Gilding: General problem: What functionality should we include in the standard library?
Bhakta: I do not see a strong motivation for this single function. Being used in a few applications is insufficient.
Svoboda: Traditionally, we would say "Get this added to glibc or some other platform's standard library first".
Bhakta: That was my position on the last paper, but clearly WG14 has different priorities.
Voutilainen: This could be a family of functions, that would be more worthwhile than a single function. I do not think that a general function would suffice.
Myers: Some features seem straightforward and can be judged based on proposed text, while others seem likely to have issues that come up in implementation. That Is why implementation experience is important.
Keaton: I propose that we have a poll to thank the author, and encourage him to get it added to two of the existing libraries for future consideration by WG14.
Ballman: I would be more in favor if this was a family of algorithms.
Gilding: I would set the bar lower for library functions that have prior implementations. Implementation experience shows how something is to be built.
Myers: The specific proposed sample implementation suggests there is a proposed "population-count" operation as a smaller more general function and more likely to be approved by the committee.
Bhakta: I would not like to have discussion on general support on this paper. It would be good to have the thank-the-author-and-implementation-experience poll Keaton suggested earlier.
Voutilainen: This seems more like a question of whether this sort of functionality is appropriate for the standard library. I would not invite the proposer to get this into an implementation.
Bhakta: I agree. I was just trying to move it forward.
Bagnara: A good poll question would be: Does WG14 believe the proposal comes with sufficient rationale for inclusion?
Keaton: I agree. I will ask us to take that poll, and then relay the results to the author.
Opinion Poll: Does WG14 believe the proposal (N2684) comes with sufficient rationale for inclusion into the C standard? 0-20-1 Clear direction.
Action Item: *Keaton*: Notify the author of N2684 of WG14's poll.

Monday

- 5.5 Múgica, #warning directive [N 2686]

Svoboda: Why is there no normative text that distinguishes between #warning and #error?
Ballman: It is addressed by normative text, but much of that text is elsewhere in the standard so you will not see it in the paper.
Voutilainen: Have there been earlier concerns about adding #warning in the past?
Ballman: I did not see any previous proposal in the WG14 document logs. I could have missed it.
Voutilainen: Many projects use #warning. Many other projects did not use it to maintain portability.
Keaton: I think #warning is not standardized only because there has not been any proposals promoting #warning yet.
Bhakta: Lack of standardization allowed people to add #warning because it would raise a diagnostic if unsupported.
Voutilainen: Is there a WG21 proposal?
Ballman: I have not asked Múgica but I would recommend this proposal for WG21.
Voutilainen: If Múgica cannot do the C++ proposal, then Ballman and I should help.
Decision Poll: Does WG14 wish to adopt the proposal N2686 into C23? 17-0-1 It goes into C23.

- 5.6 Múgica, Identifier - primary expression [N 2687]

Gustedt: I think the definition of "primary expression" is sufficiently clear. On the reflector, I proposed to interchange the two paragraphs so "constants-is-primary-expression" comes first.
Gilding: Moving footnote 1.3.2 to not be a footnote should have no effect. Based on experience this will change the standard for the better & make it more searchable. So this is a teach-ability improvement.
Bhakta: I find the new wording worse (e.g. more confusing, less clear) overall in both sub-proposals. I agree in principle that original wording is unclear. The footnote would be much better totally removed.
Ballman: Question: Is that because this is in a Semantics section? Do we have any normative requirement that a failed lookup requires a constraint violation, or is it always undefined behavior? (pause) The fact that no one provided an immediate answer suggests that the standard does not make this clear.
Bhakta: I would guess that it is implicit undefined behavior, but I am not sure.
Gustedt: I am not sure. A syntax violation is stronger than a constraint, if this is a syntax error.
Ballman: I personally assumed that violation of syntax is not undefined behavior, but should produce an error.
Gilding: By definition, a syntax violation must be the strongest kind of violation. Missing lookup should be implementation-defined. If a translation can assign a meaning to a missing token, that is considered an extension.
Yodaiken: What is the advantage of allowing compilers to accept syntax errors? What do we enable by calling this undefined

behavior?
Bhakta: We have a global table lookup that allows you to learn special aspects of a thread…perhaps that addresses Yodaiken's question.
Gilding: That is what the __ (builtin) name-space is for. They are declared by compiler magic.
Yodaiken: Allowing the compiler to treat this as undefined behavior makes C harder to write & debug. The syntax should be something the compiler accepts or rejects. It should be check-able.
Myers: It is definitely an improvement. Should we examine how well we specify disambiguation in syntax productions, where you have to examine other texts to disambiguate some programs?
Ballman: That is not a bad idea. I was surprised at how little we talk about lookup when I first joined WG14 (from C++).
Bhakta: The standard could definitely be worded more clearly, and I agree with Yodaiken. I still think the wording here is not good enough.
Ballman: I have not heard anyone argue that the current wording is clear. Keaton, this is really a clarification request, how should we proceed with this paper? Something should happen, but who should handle it?
Keaton: We need a volunteer to champion a new resolution, either wording or rationale.
Seacord: This might be a "perfection fallacy"…our wording is good, but not perfect. Perhaps we should adopt this proposal now and welcome further refinements to the wording?
Ballman: That sounds like a good idea; we do not want to drop minor improvements because we have not perfected them. Did the committee not dedicate time to improve wordings in the C17 era?
Keaton: That was true then, but now with C23 imminent, we should not stack up things to do later.
Ballman: Once we bring that process back, can we not drop it again?
Keaton: We still need to discuss whether we want to bring back the process. There have been suggestions of publishing new editions every three years, this should get folded into that plan.
Gilding: Should we consider Gustedt's suggestion (e.g. proposed wording #3) of swapping the paragraphs?
Ballman: I would prefer someone championed that proposal with a new document.
Keaton: I am all for homework; let us get this done this week if we can.
Bhakta: I agree with Seacord's point of anti-perfectionism. But I still think adopting either sub-proposal is a step backwards.
Ballman: Bhakta, why do you find the new wording less understandable?
Bhakta: I will get you that later during the meeting.
Myers: We could put something in Constraints to force a diagnostic.
Ballman: Yes, there is a question of should this be undefined behavior or a constraint violation? Resolving that makes this more of a proposal than a clarification request.
Yodaiken: What's wrong with "this is a syntax violation"?
Ballman: When constraints are listed inside the "Semantics Section", violation is traditionally undefined behavior, not a constraint violation.
Yodaiken: Is that documented somewhere, or just a convention?
Gustedt: Only things that appear in a Constraints section are constraint violations.
Yodaiken: But a syntax error is not a constraint violation, is it?
Gustedt: Good question.
Ballman: I do not think "syntax violation" is addressed anywhere in the standard.
Myers: s5.1.1.3p1 mandates compilers issue errors on constraint or syntax violation.
Goldblatt: There seems to be widespread agreement on what the semantics ought to be. Can we have a wording expert champion it?
Gustedt: Being a footnote is weird, it says this would be a violation of the syntax, but this is not clear from normative text. Being in the text is weird, because it is in the Semantics section; it should be in a Syntax section.
Jabot: In the Primary Expression section, every sub-clause has a Constraints section except that one.
Myers: Although we have Syntax sections, at least some syntactic disambiguation rules do go in the Semantic sections.
Ballman: Is anyone willing to help Múgica with fixing the wording on this proposal? (none) No volunteers. Do we give Múgica feedback & hope?
Gilding: I do not mind volunteering, but I am not the biggest expert here. The standard is very implementation-led. I have no suggestions what the solution to that would be.
Yodaiken: I still do not understand what is confusing about the proposed wording?
Bhakta: I said it is less clear than original wording. In the wording of the enumeration constant; the two proposals are saying the opposite things (with regard to it being a primary expression).
Myers: Perhaps we should have some straw polls for feedback?
Keaton: It is unambiguous right now. An undeclared identifier is a syntax violation, not a primary expression. If people want clearer wording, let us get that done this week, but Múgica cannot help us this week.
Gilding: What is the point of having undefined behavior here?
Wiedijk: Keaton says it is correct as is now. Enumeration constants are identifiers as well, but they are not so declared. So I understand why enumeration constants are confusing. For me, no change is needed.
Keaton: It is not necessary to have undefined behavior to allow extensions. You can allow syntax not defined by the standard. A syntax error is not undefined behavior. Do not use undefined behavior just to allow extensions. Can we get enough clarity here on what is needed, so that someone will work this week and we can re-examine later in the week.

Ballman: Gilding volunteered and I can help.
Gilding: It sounds like the biggest question is: Where do we want the violation to be? We can also clarify what the intent should be?
Opinion Poll: Does WG14 want to see an explicit statement that an undeclared identifier is a violation of the syntax for primary expressions? 13-0-8 Clear direction, given that we have no words yet
Keaton: OK, we will come back to this once Gilding & Ballman have different wording. Thank you Gilding & Ballman!

- 5.7 Múgica, Sterile characters [N 2688]

Bhakta: I cannot speak to intent, but practice: In IBM's case, you can have different results in the execution character set based on context.
Gilding: This feels like reading a stream-register device. The leeway should exist.
Ballman: I am hearing a preference for proposed wording 1 over 2. Does anyone wish to argue for porpoised wording 2? (none)
Ballman: Do we want to define a term of art for sterile characters? e.g. wording 1.b but replacing "non-encode-able character" with "sterile character".
Bhakta: There is a little more than the intent.
Ballman: Yes, we add the implementations tell the users. Does anyone find that too burdensome? (none)
Decision Poll: Would WG14 like to see N2688 Proposed Wording 1 adopted into C23? 15-0-5 So it goes into C23.

- 5.8 Goldblatt, Sized Memory Deallocation [N 2699]

Svoboda: What happens if size is incorrect in free(size)? Undefined behavior?
Goldblatt: There is a pre-condition that the size is correct. Most implementations have some degree of checking. So they can verify & abort on pre-condition. Or we can put in thread-local cache without checking. So you could get weird results. All allocators can provide security checking.
Svoboda: I would prefer that undefined behavior be explicit, or implementation-defined. Do platforms that support free_size() also support free() (without size)?
Goldblatt: Yes
Gilding: So this can actively improve the security of programs by checking size. That is a really good improvement. I feel like there is an overlap with users who ask malloc() for a huge chunk of memory which they then do their own alloc/deallocate on. Would there be use in requesting a memory pool (e.g. a new malloc())? This would allow users to express their own intentions.
Goldblatt: It is something we have played with in our implementation. It is a chicken-and-egg thing getting to a stable API. We cannot provide much design guidance. But the security claim is not theoretical. In our debug mode (at Facebook) we found a few bugs & our Google co-authors discovered the same thing.
Wiedijk: There is a wording issue: "Otherwise the behavior is undefined", should be "result" instead.
Goldblatt: There is another wording issue: free_size(null), that is not the intent.
Bhakta: Here is a more negative view of Gilding's comments: The parameter does increase the range of undefined behavior allowed now. So this could help security, or it could cause more problems.
Goldblatt: I am personally interested in this proposal more for performance benefits than security benefits.
Bhakta: This also doubles C's memory API. Is there another reason for standardizing this?
Goldblatt: There are two benefits: one to users who want portability. There is currently a non-trivial overhead for determining if a platform supports free_size().
Seacord: Several years ago, we proposed a config_allocator() function about how to manage memory before calling malloc(). Passing the pointer to this new function: It is a good idea to set a pointer to null after calling free on it. It would be nice if free() nulls out the pointer. If free_size() took the address to the pointer, it could set the pointer to null. That would require a change to the interface.
Goldblatt: I am weakly opposed to that, because it introduces a less-efficient calling convention to that function.
Steenberg: I often wrap in my code. Wrapping free() to free_size() is trivial. I am not opposed to it, but it seems trivial for platform-specific code. If someone gives me a pointer I cannot tell if it is valid. Some kind of introspection would be useful.
Goldblatt: If you get a heap pointer but do not know its size, you can still free() that pointer.
Steenberg: The argument that standardizing this simplifies code is rather weak, because it is easy to wrap free_size().
Goldblatt: There was an attempt to add this to OpenSSL by my co-author. You do not want to increase platform-specific dependencies.
Voutilainen: When these functions are available in standard C, some users will switch from free() to free_size(), right?
Goldblatt: Yes. I did not emphasize the performance benefits. In C++ case, more time is spent in an allocator than in C. We saw a 1% end-to-end performance improvements. This is a fairly substantial performance-impacting change.
Krause: Should we reserve all free_* things in the future?
Goldblatt: That would be reasonable. I do not see that there is lot of new variants of free().
Ballman: There is one downside to reserving it: I suspect people have many existing API's that start with "free".
Goldblatt: These names are safe with regard to OSS projects, according to Google.
Gustedt: Since we have the "mem" prefix, should we use it?
Goldblatt: I am not wedded to any particular name. We can poll on names.

Gilding: You end up with a vicious cycle of name spaces if you use name spaces starting with current standardized functions.
Goldblatt: The naming was the part of the design where I most wanted committee input.
Bhakta: You have already mentioned aligned_size_free() in the paper; was there more discussion about whether we need an "aligned free" function?
Goldblatt: You do need the aligned bit as an implementation constraint. One suggestion is to only have aligned_size_free(), which behaves like size_free() if you pass 0 as the alignment. I find that to be a messy API.
Opinion Poll: Would WG14 like to see something along the lines of N2699 adopted into C23? 16-0-6 Clear direction that we do want this.
Seacord: I recommend a poll on existing names.
Opinion Poll: Would WG14 prefer the existing names "free_size()" and "free_aligned_size()" over other names in N2699? 13-4-5 Pretty clear direction that people want the existing names
Wiedijk: Is the issue about calloc() overflow still at large? What happens if its arguments overflow? If we vote this in, I prefer the "force" option.
Goldblatt: I feel strongly that we should do the third option. But I mostly regard this as orthogonal to the content of the paper.
Seacord: It is not really required for this paper. I would prefer rewording calloc(() to mention that if the product overflows, calloc() should return NULL. There are links and the BadAlloc vulnerability where calloc() implementations did not check for wrap-around.
Keaton: Are you volunteering to submit a paper for that?
Seacord: OK
Action Item: *Seacord*: Submit a paper standardizing that if the arguments to calloc() wrap when multiplied, calloc() shall return NULL.
Myers: I also support option 3.
Goldblatt: Does anyone disagree that this content is orthogonal?
Wiedijk: The wording is there that it is a product…is that a "C" product?
Goldblatt: Yes. This issue does not affect the semantics of any existing implementations (except those that we consider buggy).
Gustedt: I think you want the opposite here. There would be several possibilities to match to size.
Goldblatt: If we merge this paper into C23, then if some implementation allowed overflowing arguments to calloc() to return a non-null pointer, that implementation would have to accommodate that during free_size(). I think that is a fine burden to impose on implementations.
Seacord: I am just going to make the behavior explicit in calloc(), if the product wraps, calloc() must return NULL. In that case, the current free_aligned size is fine, no changes necessary.
Bhakta: I will make an exception, but I normally do not like to vote on words not in a paper.
Keaton: That is a good point. The words are in front of us, but not in a single paper.
Gustedt: I would vote no for three reasons: First, the words are not in a paper. Also I am not comfortable with the naming issue, and finally, I am not comfortable with delegating calloc() to another paper and not having that paper.
Goldblatt: OK, I will come back with this later.
Keaton: For anyone who is doing homework this week, please ask me for a document number. Normally you would ask Dan.
Seacord: Do you want my calloc() paper as homework or for a subsequent meeting?
Keaton: Subsequent meeting…we have not discussed it yet. But if you want to do it as homework, we can look at it later in the week.

- 5.9 Krause, No function declarators without prototypes [N 2773]

Myers: It is a good idea, but we should give the syntax the same meaning as in C++.
Bhakta: I strongly disagree with this paper. There are many places, and we are trying to pull something away that is very useful for certain environments. You're allowed to take any number of parameters, which is useful for variadic functions. Also you have C calling assembly or vice-versa. So it is useful to have callable, but non-prototyped functions. C is cross-language and used to talk to many other languages, So this is too useful. The concrete example on MVS with regard to variable arguments is having a high-order bit being set on. But you cannot declare this in C code. So we are talking about removing useful functionality which is a weak argument for making the language cleaner.
Krause: But if it is that useful, why has the feature remained obsolescent for 33 years?
Bhakta: Mainly because I have not submitted a paper. That will change from now on. Perhaps we should rename this from "obsolescent" to "not recommended practice".
Svoboda: Does this mean that "int foo();" stops compiling?
Krause: Yes
Svoboda: I still see this often; it is too useful to eliminate. But it should not be dubbed "obsolescent', but rather something like "not recommended".
Gilding: This is still used. Perhaps we should provide a keyword for allowing code with this to compile?
Bachmann: I would rather allow variadic functions without a first-argument of fixed type.
Ballman: This option has been deprecated for longer than some of our committee members have been alive. If there is a strong sentiment to not remove this feature, someone needs to write a paper to un-deprecate it. It comes up all the time, and people are shocked that C behaves differently than C++. It is a real pain point, and there are more users that get bitten by C's current behavior.

Meneide: Such functions undergo the same kind of default argument promotion as variadic arguments. So we could eliminate the requiremen that variadic functions have at least one argument.
Myers: Allowing variadic 0-argument functions (which C++ supports): maybe some of the C++ people can comment on how that is intended to be used in C++, vs. how it might be used in C?
Gilding: It sounds like the 1-argument requirement for variadic functions comes from one historical implementation.
Gustedt: I am uncomfortable with the paper without a suitable replacement as Bhakta mentioned. If you want declarations of functions that can be implemented with C, there is only one mechanism (in <varargs.h>) to obtain … arguments.
Ballman: How long must something remain deprecated before we can remove it?
Keaton: Ballman, I do not think we can have a fixed length of time. It is possible something can be deprecated and used because there is no other way to use it. It is a C++ liaison issue that ought to be fixed.
Meneide: I studied the history of why variadic functions required a fixed argument. Gilding was right, a formalization of what one implementation had to do. Every other platform could use 0 fixed arguments. So they standardized the fixed argument. Many platforms like MSVC allow versions with 0 fixed arguments.
Yodaiken: What happens to Bhakta's code if people use it in an obsolete way? Is there a warning or a syntax error?
Krause: STDC still uses the address of the first argument, but that could change. We want to use this to call assembly code. Our users interface to assembly functions would not be happy with C function declarations without prototypes. They want to pass types smaller than int or double. Bhakta, is that a problem for your users?
Bhakta: In our calling convention for assembly, every argument is passed as an address, so it is not affected by promotions.
Svoboda: Variadic arguments do change argument sizes (because of the lack of type safety). Passing NULL as a variadic argument can screw up typing. So using … for functions is not a clean conversion.
Voutilainen: The usage of variadic functions is defined and called in C++. Also, this is a common quiz question for students; they still claim that "int foo();" is a function that takes no arguments, and are shocked to learn otherwise. Perhaps it would be more reasonable to move to C++'s practice.
Sebor: When a feature is removed, implementations do not stop supporting it, they typically issue a diagnostic warning. To the concern this might introduce confusion that does not exist today, this is how all implementations work. So I support the removal.
Gustedt: This is a decision we have to take (should this be a constraint violation?)

Opinion Poll: Would WG14 prefer to do with argument-less functions?

1. Leave as is as obsolescent feature (that is, status quo) 7-12-5 People do not want status quo
2. Keep the feature and make it non-obsolescent? 9-11-4 Not enough sentiment to change to this option
3. Make it a constraint violation. 5-15-4 No sentiment to proceed
4. Change the semantics to match C++ 16-6-2 Clear sentiment to proceed

Steenberg: I am fine with moving to C++, but still need interoperability with other languages…how do these two priorities interact?
Krause: If we do not want to introduce variadic arguments, we can go by casting function pointers.
Gilding: Variadic argument list and unspecified argument list are not the same. Removing that restriction on va_arguments would produce the exact same functionality.
Svoboda: I made that comment, see CERT rule DCL11-C for more info. I could be wrong here, but I would want to see research or experience in converting argument-less functions to variadic functions before approving such a proposal.
Krause: Variadic arguments could have a different ABI, so you could get a difference there.

- 5.10 Krause, Sane C library, when wanted [N 2720]

Steenberg: We should have a version number.
Wiedijk: "SANE" is the name of a scanner library. I would prefer a different name.
Bhakta: "SANE" has big connotations; we would expect a different name than "SANE". Some discussion before was about using different functions. Are you discussing using the WANT macro to tweak function signatures?
Krause: The idea was not to use new function names. One would have declared these functions to return const char* if there were no conflicting implementations.
Gilding: Could it use qualifying changes that do not modify the ABI?
Krause: I think C would allow these things to affect the ABI.

Tuesday

- 5.11 A Provenance-aware Memory Object Model for C (3 hours)
  - 5.11.1 TS 6010 continuing discussions (previous working draft for reference [N 2676]) (1.5 hours)

Bhakta: With regard to types that have no trap representations: We agreed signaling NaN's are floats or doubles but are not trap representations, right? What do you want?
Sewell: I do not care about that now. I want to focus on types with no trap representations.
Yodaiken: I do not see any utility of making Case 2 undefined behavior.

Gilding: Some of the committee would prefer to change it based on avoiding undefined behavior.
Svoboda: In the June meeting minutes, we had a straw poll: For reads of scalar non-character types of uninitialized automatic storage duration variables, should the semantics be independent of whether the address of the variable is taken? 14-3-4
Sewell: Yes, but that was followed by several inconclusive straw polls.
Bhakta: We should either redo the straw poll, or move forward.
Keaton: In addition to the optimization issue, this allows platforms to have tagged registers that trap on uninitialized read.
Bhakta: We have had responses from compiler optimization people. You do get feedback from them. Sebor has added comments. Not sure how that helps.
Wiedijk: You can read a value, but it is an error value and is undefined behavior. Which answer is that in your proposed straw poll?
Sewell: Some kind of wobbly value
Gilding: We should hear from people with a greater understanding of what these changes would be downstream.. Once you have read a value and stored it elsewhere, it should be stable.
Yodaiken: I think error semantics would be correct. But undefined behavior would be very wrong semantics. And I am against wobbly values.
Sebor: Everyone has their own favorite option. Implementations are all over the map. WG14's job is to standardize existing practice, but there is no consistent existing practice. The implementations will take a while to change if WG14 mandates some specific behavior. Compilers with aggressive optimizers also tend to provide the most options to allow developers to tweak the compiler's behavior.

Sewell: I would like there to be a range of options so that users can pick what they like. I do not think compilers provide a clear set of well-defined options for these things. Is there a GCC switch that says "give me option A"? There are other choices (e.g. zero-initialization). I want to conduct this straw poll: For an uninitialized read of a scalar non-character typed object of automatic storage duration, for a type that does not (on the platform in question) have trap representations, if the address is taken, should it be:
a. an allocation-time nondeterministic choice of a concrete value (stable if re-read)
b. some flavor of wobbly value
c. some error semantics

Sebor: In GCC, the "option A" switch is under review as we speak. It comes from the same authors as the Clang implementation. That is why I am arguing for doing nothing. There is no clear consensus among compilers.
Sewell: You could argue that the committee has neglected this for decades. We should specify, possibly, multiple options.
Voutilainen: Question for *Ballman*: Has the C++ liaison studied this material?
Ballman: SG22 has not considered this material yet. I hope it goes to SG1 first. The WG21 undefined behavior SG has seen this in an earlier form. SG12 was quite positive for this work.
Voutilainen: Sewell: How much effort did you put into making these semantics the same for C & C++?
Sewell: That is a difficult question to answer. I would desire it to be uniform, but I lack cycles to apply it to C++.
Krause: Someone is opposed to wobbly values because they make it impossible to write reliable programs. With regard to option C, there can be other semantics besides undefined behavior.
Bhakta: I think that last time you did a straw poll about Option C and came to no consensus. Why is it there again?
Sewell: Because the other options have changed.
Steenberg: I like Option A but also with the capability to throw out errors. MSVC gives a non-wobbly value, but in debug mode, it gives a diagnostic.
Bachmann: Representing the users, I strongly prefer Option A.
Yodaiken: There is a previous note: if you have wobbly values, do not use uninitialized values for reliable code.
Sewell: Compare the guarantees that a programmer gets from wobbly values vs. other undefined behavior.
Yodaiken: You're asking me to choose between two obscene alternatives. Sebor suggested that the Clang option is being imported into GCC. Clang can force initialization of all variables. If aggressive implementations offer users a choice, between initializing all memory or aggressive undefined behavior, I do not think that is particularly useful.
Svoboda: When we do the straw poll, please paste it in the chat. Also, I personally prefer Option C2 (diagnostic) but not Option C1 (undefined behavior). Probably we should do a yes/no on each option. Or break Option C up into separate choices
Keaton: Yes, breaking up the poll is a reasonable way to go.
Uecker: The error semantics, while nice, might not be implementable, because you have to track memory state at run-time.
Sewell: Misinterpretation of the question: "a diagnosed compile-time or run-time error at the implementationementor's choice". If they fail, you still get something stronger than undefined behavior.
Uecker: I like Option A because it contains as locally.
Keaton: I was intrigued when you suggested more than one option in the TS. That might be an elegant walkway to address Sebor's concern. I would encourage you to make it implementation-defined which options a platform uses.
Bhakta: All compilers have auto-initialized option. All users have chosen that this is not important to them.
Sewell: That option is not on this list.
Bhakta: Our users have not come to us and said "We want a fixed-value or error". What compilers have is what is satisfactory to our users. We have done what we see our users wanting. We should expend energy on what our users have not requested.

Sewell: That is a misinterpretation of what your users want.
Bhakta: Our compiler gives Option A for debug mode. At higher optimizations, you get Option C1 and wobbly values. You can get partial compile-time messages with various debug modes.
Sewell: Why Option C1 rather than Option B?
Bhakta: Undefined behavior goes into nitty-gritty about optimization.
Sebor: I am not sure I understand the options in the poll. There are two alternatives being offered: either unspecified behavior or undefined behavior.
Sewell: We have failed to understand this when using the wording in the standard. We are now trying a theoretical "what should the standard require". Please ignore what it currently requires.
Keaton: The purpose of this TS is to publish something for people to criticize. It might go into the standard or not, but it should spur discussion & get experience on it. This is not a final standardization; it is rather an experiment.
Voutilainen: To Bhakta & *Sebor*: I am skeptical on how much feedback platforms get. Not getting reports does not mean there is no problems or desire. Very few programmers know to bug their implementation vendors about extensions. Question for *Sebor*: If we nail down these semantics, how much of a philosophical problem is it for you to provide flags that are non-conforming?
Sebor: GCC tries to be a conforming compiler. There are options that go beyond conformance. I do not recall one being introduced recently. Those are historical options on their way out.
Svoboda: Question: What is Option C1 (plain undefined behavior) contain outside of wobbly values & traps?
Sewell: Option C1 means undefined behavior but not restricted to wobbly value or trap. (It could do both, for example).

Opinion Poll: For an uninitialized read of a scalar non-character typed object of automatic storage duration, for a type that does not (on the platform in question) have trap representations, if the address is taken, which of the following should be reasonable options? Bearing in mind that if there are several good candidates, the TS might identify a set of them that compilers could adopt, not just a single option.
A. an allocation-time nondeterministic choice of a concrete value (stable if re-read) 18-3-3 5 prefer this
B. some flavor of wobbly value 12-10-1 1 prefer this
some error semantics:
C1 plain undefined behavior 9-12-2 7 prefer this
C2 diagnosed compile-time or run-time error or Option A (at the implementation's per-instance choice) 16-5-2 9 prefer this
C3 diagnosed compile-time or run-time error or some flavor of wobbly value (at the implementation's per-instance choice) 9-14-0 2 prefer this
D. some other semantics (or set of semantic options) 2-12-9 0 prefer this

Voutilainen: Getting single consensus is unrealistic to me.
Sewell: I agree. The TS will probably contain multiple options. There will be no Right Thing, but instead multiple WG14-acceptable things.
Sebor: If the TS provides a set of choices for a platform, I do not see how that set of choices is anything but undefined behavior.
Sewell: The standard can say it is implementation-defined for a platform to decide that undefined behavior is one of the choices.
Sebor: That would be a radical departure from historical practice.
Keaton: It would be different from anything we now do, but it is trying to address the fluid environment we have right now. Just picking one does not address the current environment.
Sebor: If the C standard introduces this bifurcation (where "implementation-defined behavior" includes "undefined behavior"), that affects the C++ standard and the POSIX standard. If we want to discuss this, that requires a more general discussion (more than just about memory safety).
Gustedt: We could make this part conditionally normative.
Keaton: Sebor objects to undefined behavior being part of implementation-defined behavior.

Opinion Poll: For an uninitialized read of a scalar non-character typed object of automatic storage duration, for a type that does not (on the platform in question) have trap representations, if the address is taken, which of the following should be reasonable options? Bearing in mind that if there are several good candidates, the TS might identify a set of them that compilers could adopt, not just a single option.
This time, assume that the address is never taken.

1. The same as the semantics chosen for the address-taken case 11 prefer this

A. an allocation-time nondeterministic choice of a concrete value (stable if re-read) 0 prefer this
B. some flavor of wobbly value 0 prefer this
some error semantics:
C1 plain undefined behavior 5 prefer this
C2 diagnosed compile-time or run-time error or Option A (at the implementation's per-instance choice) 7 prefer this
C3 diagnosed compile-time or run-time error or some flavor of wobbly value (at the implementation's per-instance choice) 0

prefer this
D. some other semantics (or set of semantic options) 0 prefer this

- 5.11.2 Gustedt, Enforce storage stability [N 2756]

Ballman: WG21 defines "value-initialized" differently.
Tydeman: What about a struct with a flexible array member?
Gustedt: Yes, a flexible array member would be where the size is not known in advance. Writing into some part does not mean the rest is initialized.
Myers: We need to discuss two compiler areas: My concern is cases where compiler transformations mean padding bits may not exist in certain contexts. A struct with a scalar is supplied and padding bits do not exist at all. Consider: x86 extended-precision long double format has 10 value bits, which fits with padding bits in memory. But when storing it in a floating-point register, the padding bits do not exist. Do these rules preserve those padding bits? This may not be an issue in practice.
Gustedt: This could possibly be problematic for platforms that represent a struct or small array by its different components by different members.
Bhakta: One case you mentioned was when reading an object twice, it cannot change value. Do not forget about cases where you do expect a different value, such as signaling NaN's.
Gustedt: Yes, I have not mastered signaling NaN's.
Sebor: This is a proposal not for WG14, but for the Memory Object Model SG, correct?
Gustedt: Yes, but that SG is part of WG14, and we want consensus for 6010.
Sebor: Does the lack of consensus in today's straw polls mean no consensus for this?
Gustedt: That could be. But I am focusing on partially-initialized objects, which is a different direction than the previous proposal.
Sebor: To what extent does this reflect existing practice? As opposed to what we would like to see in future implementations?
Gustedt: I tested this on the implementations available on Godbolt. All things I tried for the first few platforms came up fine.
Sebor: If an implementation claimed conformance to the proposal, what is its minimum effort? Does the macro provide indication to that implementation's conformance?
Gustedt: The macro is for implementations that have difficulties to conform, because they will have different values if you access the page the first time. If the macro is 0 or undefined, then all sizes conform.
Yodaiken: If something is referenced through a volatile pointer, what are the implications for that?
Gustedt: I am not sure for the initialization part. I would stick to the current semantics for "volatile".
Wiedijk: Does an increment operation initialize its variable?
Gustedt: If the variable was uninitialized before, then that constitutes an uninitialized read.

- 5.11.3 Gustedt, Add annotations for unreachable control flow [N 2757] (for both C23 and TS 6010)

Krause: "unreachable" would pollute the name space. It needs to be __unreachable and included by header. It allows optimizations and diagnostics at compile time. Should it give a run-time diagnostic too?
Gustedt: This is in the paper…there are many more items in the paper than I have been able to cover here.

- 5.11.4 Uecker, Indeterminate Values and Trap Representations, v2 [N 2772]

Bhakta: In Section 3.1 in the paper, even with the uses of indeterminate state, it seems ambiguous talking about values. This makes it less clear than our current wording.
Gustedt: What term would you suggest?
Bhakta: I would suggest using "indeterminate representation" by itself.
Uecker: We use "state" because it is already used in the standard, e.g. the Atomics section
Bhakta: If the object goes out of scope…
Uecker: …Then it is indeterminate, and referring to it is undefined behavior. Pointers to it become indeterminate. I am not opposed to changing it to another term.
Bhakta: In the change to s6.2.4, in section 3.2: "storage duration of objects": The value of a pointer becomes "invalid". The wording is not great. The term "invalid" implies a trap, but you are not trying to say it is a trap here.
Gustedt: We already apply that terminology in the standard.
Bhakta: I had read "invalid" in other places too, but OK. Another wording nit: The pointer has a value, it does not "store" such a value.
Uecker: To me "store" and "have" mean the same thing here.
Sebor: The term "indeterminate value" has been with us since the first introduction of the standard, and it is being used in other standards. Introducing a slightly different term is not helpful, without making corresponding change in other standards (C++, POSIX), and possibly other languages that have adopted the term "indeterminate value" in their standards. Next, replacing "indeterminate pointer" with "invalid pointer" prevents an implementation from setting it to 0, which is a common thing to do, at least in GCC.
Uecker: C++ has a different definition of "indeterminate value", if we keep ours, the two standards will have incompatible

definitions.
Sebor: A null pointer is a valid pointer value, so invalid values cannot be set to NULL.
Uecker: In C++ you can copy an indeterminate value from one place to another, but not in C. But we could make usage of past-the-life pointers into undefined behavior.
Yodaiken: A paper that discusses existing practice of used pointers to out-of-scope objects: How does that fit with this paper?
Uecker: The idea was not to make any semantic changes.
McKenney: This will be discussed in SG21. We did bring in N2369 which tried to outlaw pointer zap. So we have been working to find something that people can live with. The difference between N2369 and the proposal in intomorrow's talk is, you have to add to your program when pointer zap could be forgiven.
Steenberg: I think the name changes are good. Even calling it a "non-value" is a good idea.
Uecker: For the proposal for Annex L, could an Annex L expert verify the proposed changes to the "Alternative 3" to Annex L?

Wednesday

- 5.12 Tydeman, Overlooked SNAN wording changes [N 2710]

  Bhakta: CFP wants this. We thought this was editorial, but got no responses. How should we handle this procedurally?
  Keaton: Are you saying this is editorial because it is in Annex J?
  Bhakta: The changes of prefix SNAN with a type was done several meetings ago, but the changes were not done everywhere. This paper comes because of the lack of response.
  Myers: The editor can decide whether to accept a merge or require a new paper.
  Keaton: You can make a paper by creating an issue in Gitlab. The editor checks all issues. Or you can make a merge request. Please make this correction when submitting to the editors.
  Decision Poll: Would WG14 like to adopt N2710, with the change "LDL SNAN" to "LDBL SNAN", into C23? 16-0-2 So it goes into C23

- 5.13 Tydeman, fmin, fmax [N 2711]

  Myers: The <tgmath.h>. You're proposing an example, not to remove anything?
  Tydeman: Correct.
  Gustedt: There is one occurrence of f_maximum?
  Tydeman: I believe there are 4 functions now replacing 2. Paragraph 15 is about tgmath stuff. Section G.5.1 is correct because the NaN's have been taken care of. What you see on the screen is correct.
  Decision Poll: Would WG14 like to adopt N2711 into C23? 11-0-6 So this goes into C23

- 5.14 Tydeman, intbool_t [N 2712]

  Banham: Caveat: If you put isalpha(..) == true, it will fail. unless you are implementing compiler magic with the new type.
  Tydeman: Such code has the same problem right now.
  Banham: Agreed. The new type is misleading. I agree the current situation is not nice. It is a case of programmers understanding the true nature of these functions. A separate type is useful in its own right.
  Krause: I do not think intbool_t adds value to these functions. They should be turned bool, but cannot due to legacy implementations. We have had a short look at the WANT macro for the library. These functions would be a good candidate for the WANT macro. The sentence that future functions should use bool…that sentence could be improved.
  Tydeman: This is strictly for functions that return bool, not success vs. failure.
  Gilding: I agree, this seems actively misleading. Maybe intbool_t can control an implementation; if defined, the definitions of these functions return 0 or 1. Something similar was brought up by N2541 to convert these to bool. That may have generated the WANT macro. Without solving the ABI issue, this cannot progress.
  Bhakta: Efficient implementations cannot do a conversion to bool…this typedef makes user code more misleading.
  Ballman: I am curious if any large C libraries have implemented anything along these lines?
  Tydeman: I do not know of any.
  Wiedijk: We have bool and intbool_t. This is duplication and I am bad at choosing.
  Gilding: Ballman: The proposal is literally just a typedef…what experience would you want? It could work with the MISRA essential type system. Adding this along with a boolean intrinsic would make type-checking less effective.
  Ballman: I am curious if anyone has implemented it, what issues did users run into? Bug reports / Confusion?
  Krause: SDCC used an 8-bit type for these functions, we changed to (16-bit) int for standards compliance. We got complaints from users about the type extensions. We added inline versions & optimizations, which made our users happy.
  Banham: We are trying to get back to C's logical operators, which used 0 or "not 0"…this is distinct from the "bool" type (0 or 1). That is where the newer approach clashes with the older. I like separating concepts out…I just object to the "intbool" type, but perhaps it could be renamed to "int_logical_t" to support the 0 or "not 0" concept.
  Myers: Some operations return "0 or nonzero"; others return "0 or 1", such as function sin() from <math.h>. You are using the same

type for two different sorts of functions.
Gilding: The objects on N2541 were limited to the <ctype.h> functions (0 vs. non-zero). Are there any objections to simplifying the 0-vs-non-zero vs. 0-vs-0 into two categories?
Tydeman: I am willing to do that.
Myers: The math.h comparisons that use 0 or 1 are like the comparison operators, which use 0 or "not 0". If you made those macros return bool, they would be different than the comparison operators. You would also have to have the comparison operators use bool.
Bhakta: I think that would have the same ABI issues.
Opinion Poll: Would WG14 like to adopt something along the lines of N2712, with a different name than "intbool_t", into C23? 6-11-2 So not sufficient consensus to do this

- 5.15 Tydeman, Integer Constant Expression [N 2713]

  Myers: It should be clear that implementations might still have syntax which requires constant expressions. Also, DR 312 is the other place where the committee previously clarified other forms of constant expressions.
  Bhakta: I like putting a DR response into the standard (assuming we agree). Why was it not put in?
  Keaton: With older DR's, we made no change, because that required a different process. There was a gradual migration from "we cannot make any change" to "we can".
  Svoboda: GCC & Clang disagree because of extensions, not standard compliance, right?
  Tydeman: The gray text is an example.
  Svoboda: Would this silently make certain arrays turn into VLA's?
  Tydeman: Yes.
  Gilding: Forcing compilers to change forces a change on the type system. Would this be better with a footnote or additional sentence?
  Gustedt: There is implementation practice that does it differently. Since constant expressions are implementation-defined, I think the status quo is good and some user code can have the arrays become VLA's, and the user code becomes invalid on these platforms.
  Tydeman: That code is definitely not portable.
  Gustedt: Right. But we are stepping on the user's feet, since this is implementation-defined.
  Meneide: This is a good change. But for people that rely on GCC/Clang behavior, they treated VLA's like stack arrays. We do not have to break anyone's ABI for this. You will wind up with the same ABI and no breakage when this change is done.
  Banham: Users expect that an array defined by a constant expression should be static, not a VLA. (It would be static in C++). So this will surprise users.
  Bachmann: This problem is not more or less efficient for VLA's. This could change sizeof operations.
  Gilding: I could not see any way this causes efficiency change, because platforms can implement VLA's however they like. The optimizer will not care about VLA's vs. static arrays.
  Decision Poll: Would WG14 like to adopt N2713 into C23? 10-2-7 This goes into C23

- 5.16 Tydeman, hypot() [N 2714]

  Decision Poll: Would WG14 like to adopt N2714, with the change of "returns NaN" to "returns a NaN", into C23? 17-0-2 So it goes in
  Bhakta: For those who abstained, do you know if this impacts your implementation? (none) It does not affect my implementation.

- 5.17 Tydeman, cr_ prefix [N 2715]

  Krause: WG14 often takes identifiers form users for FP uses. How many of these identifiers are currently reserved?
  Tydeman: I have not checked.
  Bhakta: For my implementation, all usage of "cr_" is consistent. Overall, this is a good thing. and a common prefix for math terms.
  Svoboda: We have a section for identifying reserved identifier prefixes…it should be updated with this.
  Gilding: Should this list be "potentially reserved" or "fully reserved"? Are we taking the entire name space from the user?
  Tydeman: I do not know the distinction. Is Annex J automatically generated?
  Meneide: Mostly.
  Ojeda: For research: I remember WG21 had a tool with many indices to look for usage of identifiers.
  Svoboda: I believe Mr. Ballman maintained that tool.
  Myers: These should be "potentially reserved" rather than "reserved".
  Tydeman: Section 6.4.2 defines "potentially reserved" and I guess that is the correct term.
  Keaton: Are you willing to put the "potentially" back in?
  Tydeman: Yes
  Bagnara: I found some hits of "cr_" matches from some C code, I pasted it into the chat.
  Myers: Those declarations would only conflict if they have external linkage.
  Decision Poll: Would WG14 like to adopt N2715, with the addition of "potentially" back into the phrase (that is, it should not be removed), into C23? 15-0-5 So this goes in.

- 5.18 Tydeman, Numerically equal [N 2716]

  Gustedt: The letter cases would be clearer if it said "the results compare equal" rather than "are equal". Also for the last one.
  Gilding: This seems like a good change. I do feel like a modifier to the second one "is equal' would be useful.
  Tydeman: I am not sure about "compare equal" vs. just "equal".
  Bhakta: I do not want to wordsmith this on the fly. There is no definition for "numeric equal". If there are further changes anyone wants, we can bring that back to the CFP group, or they can write their own paper.
  Bagnara: Perhaps +0 vs. -0 might be relevant here? Is +0 numerically equal to -0, but is +0 equal to -0 in the English sense? So the term 'numeric' does convey something.
  Wiedijk: Is a NaN equal to itself?
  Tydeman: No
  Myers: IEEE-754 defines four levels of values. If we want to introduce new terms here, let us follow IEEE-754.
  Bhakta: CFP did consider +0 vs. -0 and convinced ourselves that this change does not affect that.
  Tydeman: Paragraph 4's numerically equal is about quantum exponents, not signed zeroes.
  Myers: Maybe we add definitions to the standard for 'equal' and 'equivalent'.
  Bhakta: I am not opposed to that, but not on this paper or at this time. We would have to be very careful to avoid breakage.
  Gustedt: I find it confusing that the results are "equal" but have different values, and using "compare equal" would clarify that.
  Tydeman: OK, I will take this back to CFP for one more round.
  Bagnara: I just searched. These are the only occurrences I have found in C18. Removing them is no damage for the first case. For the second case, I agree with Gustedt.
  Bhakta: In terms of CFP vote, it sounds like each of the three changes can be done individually.
  Tydeman: Agreed
  Decision Poll: Would WG14 like to adopt the first and third change in N2716 into C23? 13-0-5 So that goes into C23
  Decision Poll: Would WG14 like to adopt the second change in N2716 into C23? 1-3-13 no consensus to make this change
  Bhakta: Could we have an "along the lines of" poll for the second change?
  Opinion Poll: Would WG14 like to adopt something along the lines of the second change in N2716 into C23? 11-0-6 Clear sentiment to do something along these lines

- 5.19 Thomas, C23 proposal - range error definition [N 2745]

  Keaton: 'shall' should not be in a footnote.
  Svoboda: Overflow is not defined in ISO C, and it has brought up confusion
  Tydeman: FP overflow is defined in Paragraph 5.
  Svoboda: OK. It is integer overflow that is problematic, not FP overflow.
  Ballman: I thought the original words were easier to understand.
  Tydeman: We are replacing "mathematical" with "what the implementation actually does"
  Ballman: So it is standardizing existing practice.
  Myers: The point is to avoid duplicating paragraphs 5 & 6.
  Decision Poll: Would WG14 like to adopt the change, but not the footnote, from N2745 into C23? 12-0-7 So it goes into C23.

- 5.20 Thomas, C23 proposal - overflow and underflow definitions [N 2746]

  Tydeman: This paper has been replaced by another paper that has not been published yet.
  Bhakta: We have asked Plakosh for a document number, but have not gotten one yet.
  Keaton: So we should skip this one now.

- 7.2 Thomas, C23 proposal - effects of fenv exception functions [N 2748]

  Keaton: One comment about first change: 'Should' is a normative term & does not belong in the footnote. Can we re-word that?
  Tydeman: "could"
  Keaton: That would be acceptable. You could also move the footnote to normative text.
  Ballman: There is a slight difference between "should" and "could". I feel like something recommended is more clear.
  Myers: If we move this out of a footnote, perhaps it should go into a 'Recommended Practices" section.
  Bhakta: Good idea! Can we use "should" in Recommended Practice?
  Keaton: Yes
  Opinion Poll: Would WG14 like to adopt something along the lines of change 1, but as a 'Recommended Practice" instead of a footnote, from N2748 into C23? 15-0-3 So clear sentiment to proceed
  Decision Poll: Would WG14 like to adopt changes 2 and 3 from N2748 into C23? 16-0-2 so it goes into C23

- 7.3 Thomas, C23 proposal - IEC 60559 binding [N 2749]

  Ballman: "cr_" is a commonly-used prefix in header files. Why are they fully reserved rather than "potentially"?
  Bhakta: We changed that to "potentially"…we did not drop "potentially".

Banham: Which version of IEC-60559 for C23?
Bhakta: Originally 2008, but we have updated to 2019 in our later paper. We are using 2019 now.
Decision Poll: Would WG14 like to adopt N2749 into C23? 11-0-6 So it goes into C23

- 7.4 Tydeman, static initialization of DFP zeros [N 2755]

Meneide: A . would be fine (vs. a ;)
Bhakta: First, there are some typos we assume the editor will fix. Change "implemenation" to "implementation". The footnote itself may not make complete sense. Also "with the most negative quantum exponent" should be "with the minimum quantum exponent".
Tydeman: Does "minimum" mean "close to 0" or "close to minus infinity"?
Myers: We do have N2727 which we will consider on Friday. If we accept this, we need to take account of these wording changes then.
Ballman: Wording problem "arithmetic type is floating-type". I think DFP's are an arithmetic type.
Tydeman: The fact that arithmetic type is 0 does not say which kind of 0.
Keaton: This is not a conflict.
Ballman: This is not less confusing to me.
Banham: Where we have got implementation-defined, this might be better as unspecified behavior. If all these values are numerically equivalent.
Tydeman: From an arithmetic-operation standpoint, you want all-bits-0 to be decimal-point 0.
Banham: Does it make sense to leave this to implementations to decide?
Tydeman: I thought CFP decided it was too strong to require all bits 0.
Banham: But you hurt portability if the implementation's choice is significant.
Tydeman: I am OK with mandating all-bits zero, but I do not know if the rest of the committee would agree.
Bhakta: With regard to confusion with arithmetic type. In paragraph 6.7.9 it is hard to be confused on what that means. I am not opposed to re-wording it, but I do not see that as necessary.
Meneide: With the all-bits-0 representation, you will get an all-bits-zero or you can get a zero with an arbitrary exponent. In GCC static initialization gives you all-bits-0, but stack implementation gives you arbitrary exponents.
Ballman: If we cannot nail down all-bits-0, I would prefer it be implementation-defined rather than unspecified.
Bhakta: I also prefer "implementation-defined" to allow cases like integer conversion; you want to be 0 after decimal point. In other cases, you want the full significant digits to be all zeroes. I would recommend voting this in as is. A new paper could address the implementation-defined aspect of all-bits-0.
Decision Poll: Would WG14 like to adopt N2755, changing "implemenation" to "implementation", into C23? 14-1-3 So it goes into C23

- 5.10 Revisit: Krause, Sane C library, when wanted [N 2720]

Gilding: Is it possible to have usage of the WANT macro to support different versions? If there were breaking changes, those should be separate from non-breaking changes.
Bhakta: Not all C standard libraries allow you to transparently switch between functions. In that case could not switch between different versions using WANT.
Steenberg: This feels like two proposals: One to add a flag to change standard library features, the other is the actual changes. Having flags like this is good, but they should be formalized. We could deprecate things we do not like. A good WANT-like framework would be a useful proposal in itself, even if we take nothing else from this paper.
Ballman: How does this get extended over time? If we want new changes in C26, would there be a different macro then, or should they expect new things to go under the WANT macro, or should we have separate macros for different features?
Krause: If we want multiple versions, we can address that now. I am not sure it is worthwhile having fine-grained control.
Myers: There are many different things one could consider fixes to existing libraries.
Sebor: I am in favor of fixing these signature problems, simply for type & const safety. I am not in favor of introducing a macro to control these fixes; they subset the language. The constness does not capture the constraint. By declaring a function to return const, that does not prevent the const from being cast away.
Steenberg: If we had a version flag, it could be a trade: I want the latest version, but that means I give up things not on that version.
Ballman: I am wondering about compose-ability. What happens if there is a conflict between different macro definitions in different translation units?
Krause: "const char*" vs. "char*" do not break ABI's.
Gilding: I wrote a paper whose idea was accepted; it did not hit the POSIX obstacle and created no ABI issues. It was rejected for wording changes.
Seacord: I like this idea of version macros to address these legacy functions that we are incapable of changing.
Bhakta: I disagree with Gilding's point of opt-out vs. opt-in.
Ballman: Our charter says "trust the programmer". But that goal is outdated. If we have to choose safe/secure vs. performance, we are doing people a disservice by lessening the priority of safety & security.
Sebor: POSIX exposes dozens of version macros, and I have yet to see a program that uses them correctly. On most POSIX system, the C library is part of the POSIX standard. Are we providing POSIX an opportunity to disable the safer POSIX signatures? We

should get in touch with POSIX and see how they would handle a WANT macro.
Bhakta: The safety/security vs. performance should be a separate discussion.
Seacord: I agree with Bhakta and disagree with Ballman. There are API changes that do not break legacy code. It feels like the version mechanism should be "opt-in" not "opt-out".
Ballman: The changes we are talking about here almost have to be "opt-in" to avoid breaking code. But in a vacuum, if you ask users to opt-in to something that makes their life better, users do not enable them. For that reason, Clang no longer accepts diagnostics that are off by default.
Jabot: A user must decide to use an edition of C. They fully expect breakage if they use an earlier version.
Seacord: Nowadays programmers do not manage their own builds anymore. Organizations have separate build experts that decide compiler flags. This increases the likelihood that, for an organization, these people would make inform decisions with regard to WANT macros.
Myers: We do seem willing to break code by adding keywords.
Bhakta: C users are far more conservative in trying to avoid breakage…many of them are pre-ANSI. Many compilers have extensions and ways to disable them, to allow users to avoid breaking code.
Gilding: There is a difference in noisy breakage from introducing new reserved words vs. quiet breakage from modifying API's.
Opinion Poll: Would WG14 like the WANT macro along the lines of N2720? 8-8-5 Probably not enough sentiment to require another paper

Thursday

- 5.11.3 Revisit: Gustedt, Add annotations for unreachable control flow [N 2757] (for both C23 and TS 6010)

Bhakta: I like it. The argument version is more useful.
Gustedt: Let us focus on "do we want this feature or not?"
Yodaiken: What happens if there are less than 2 arguments?
Gustedt: Undefined behavior
Yodaiken: What is the advantage for the programmer over calling abort()?
Gustedt: Yes, that is the distinction to be made. If we use abort(), the program has defined behavior. The advantage is that a developer can goose the optimizer (without invoking undefined behavior). Something like this is used in the Linux kernel. (because it cannot abort()).
Yodaiken: It seems to decrease the semantic clarity.
Gustedt: Agreed.
Gilding: An attribute would be more powerful & more flexible because you can provide your own function name, put the attribute on the function. I would prefer that this be an attribute.
Gustedt: I did not think of function attributes.
Jabot: There is a C++ proposal to do that (an attribute). But I am not sure of its current state.
Gustedt: It has been seen by some parts of WG21. The proposal author was unreachable. If we are serious about this in C, we should also propose it to WG21.
Voutilainen: The WG21 proposal status is basically parked. Someone would need to champion it. The benefit to programmer: imagine an unreachable 'default' clause in a switch case…which helps a compiler produce more efficient code. It has a semantic effect you can state.
Gustedt: It was useful for programmers to explicitly express that some code path is undefined behavior.
Myers: We should remove the suggestion of diagnosing functions where all code paths are unreachable. Along with the optimization thing, this goes hand-in-hand like better handling of uninitialized variables. It is useful for programmers to say that some item "cannot happen".
Svoboda: This intentionally performs something we tried to avoid, so it rubs me the wrong way.
Gustedt: Yes, that is its design.
Svoboda: True, so I will ignore my heebee-jeebees. Do we have implementation experience?
Gustedt: GCC/Clang/MSVC all have this feature in a function-like syntax.
Svoboda: Using it to rule out 'default' in switch seems to me best done by something more specific. So this would be useful as a TS so we can figure out what to do with it. I am still not sure how to use it.
Gilding: If it is not mandated to say what it can prove, the user can not trust it. Is it noreturn?
Wiedijk: If it is a function, it needs a return type, or perhaps it should be a macro.
Gustedt: It can be placed wherever a void expression could be placed.
Steenberg: I am very much in favor of this. It is useful and explicit.
Opinion Poll: Would WG14 want a feature similar to unreachable along the lines of N2757? 22-1-1
Gustedt: Do we want a function interface, or one that looks like a function but is its own grammar terminal?
Myers: If you have an argument, then you are having something type-generic, so maybe a macro?
Gustedt: It could be a macro, a la static_assert(). This is what I prefer.
Myers: Is it determined by the compiler, or by run time (or both?)
Keaton: We are out of time for this proposal.
Gustedt: OK, we can discuss offline.

- Homework: Ballman, __has_include (revised) N2799

  Wiedijk: In the first branch why is "have_experimental" there?
  Ballman: I am fine with removing it.
  Krause: I am fine with the example being short.
  Decision Poll: Would WG14 like to adopt N2799 as is into C23? 23-0-1 So it goes into C23

- Homework: Seacord, calloc overflow handling (N 2800)

  Krause: The first wording is good. We could always use recently-added integer safety functions to address wording.
  Bhakta: We are discussing this paper only in context of the previous paper, or adding to the standard.
  Keaton: We were planning on voting this into the standard, but if people are uncomfortable with that, we can defer the vote.
  Bhakta: I am uncomfortable with that, because this paper has not gone through the mailing process. This is not an offshoot of the previous proposal, because it fixes wording in the standard, independent of that proposal.
  Keaton: We want this direction, because it was necessary to clear this up in order to tackle the original proposal, which we will come back to this week. This was introduced in the original proposal, it is not out of the blue. But we will not vote on it today.
  Svoboda: I prefer the second wording, because integer overflow is not defined explicitly in C.
  Seacord: In the last version of the integer safety proposal, "C17 says unsigned integers overflow". The wording that got voted in strongly assumes this. If unsigned integers do not overflow, the language surrounding your paper is probably incorrect. Also the C++ standard is worse on this topic. The current C++ wording says: the result of integer arithmetic cannot be represented, it is undefined behavior, which includes unsigned wraparound. So it would be worthwhile for us to have a separate definition. I am surprised that no one ventured to answer my question about this.
  Svoboda: It sounds like we need a wording cleanup.
  Seacord: It sounds like you are volunteering.
  Svoboda: OK, if you review it.
  Action Item: *Svoboda*: Write a proposal to clean up C standard on wording on integer overflow (especially for unsigned integers)
  Wiedijk: The second wording needs "mathematical product".
  Seacord: If you just have a mathematical expression in the standard, that was implied.
  Keaton: I interpreted it to mean what is computed.
  Gustedt: The second version (N2801) of the proposal for the free functions makes this unnecessary anymore.
  Tydeman: The C standard said "A computation involving unsigned integers can never overflow".
  Seacord: Ballman convinced me of the opposite. I would like some clarification on the terminology today.
  Svoboda: Should the definition of "overflow" in the C standard allow unsigned integers to overflow?
  Bhakta: You are looking for changing the definition of "overflow".
  Pygott: Do we need the word "unsigned"? It can apply to "signed"?
  Seacord: Everyone knows that signed integer operations overflow. The question is, do unsigned integers overflow? This poll will guide Svoboda's forthcoming paper.
  Svoboda: The poll question is now: If the mathematical result of an operation on two unsigned integers is outside the range of the resulting type, should that constitute "overflow" as the term should be used in the standard?
  Myers: We need to review the >60 cases of overflow uses in the standard.
  Seacord: Yes, but most of those are floating-point overflow. And some are pointer arithmetic overflow.
  Gilding: This is almost impossible to vote on. You are voting on the definition of (integer) overflow, right?
  Seacord: 100% correct!
  Wiedijk: Looking at Wikipedia's pager on "integer overflow". Wikipedia is not saying that unsigned integers cannot overflow.
  Seacord: Yes, the Committee has discussed this, and discussed unsigned integer overflow, most recently in the context of Svoboda's recent papers.
  Opinion Poll: If the mathematical result of an operation on two unsigned integers is outside the range of the resulting type, should that constitute "overflow" as the term should be used in the standard? 6-6-10 No sentiment to change the definition of "overflow".
  Seacord: The direction I received is that the first wording is incorrect. Perhaps we can use "wraps", or we can use the second wording, clarifying that we are talking about the mathematical result.
  Goldblatt: I abstained, not because I have no opinion, but I want to be convinced by a future paper.
  Bhakta: I think we made a mistake in mentioning "unsigned integer overflow". Section 2.3.2 gives you what they consider overflow to be.
  Wiedijk: If it is overflowing, then calloc() should be allowed to give you that memory even though it is usable?
  Seacord: I want a voting on the wording that it be a null pointer. (e.g. that calloc() cannot succeed if the product wraps).
  Krause: With regard to Gustedt's opinion that calloc() should be able to return memory, I thought the sizeof() operator could not return the size of such an object? Can we see what the C99 Rationale says about this?
  Keaton: I will find it.
  Myers: With regard to the question of very large objects that will not fit in size_t: It is problematic to allow such objects to be created.
  Gustedt: I did not mean applications should allow such an object to be created. If we do changes for calloc(), we should make the standard clear. That is why I was not in favor to vote this in as is now. I would like to see more discussion (here, or in the paper).

Wiedijk: I do not see any problem with large objects if you use a Turing machine. But otherwise I am strongly against that kind of thing.
Gilding: I think calloc() should return NULL even if there is enough memory.
Krause: It is not possible to have any objects that are bigger than size_t.
Opinion Poll: Does the committee favor something along the lines of the second wording in N2800? 18-0-3 Clear sentiment to go in that direction.
Pygott: We were making sure that the nmem * size represented the mathematical expression, not the C expression.
Seacord: Agreed.

- Homework: Goldblatt, Wording for Sized Memory Deallocation (N2801)

Wiedijk: There are functions that internally call malloc(). The behavior might be implementation-defined rather than undefined, if an implementation documents what size it uses for these functions.
Goldblatt: This wording is fine if a function calls malloc(strlen(s)+1)
Decision Poll: Would WG14 like to adopt wording candidate 2 in N2801 to be adopted into C23? 17-0-5 So it goes into C23

- 5.21 Boehm, Clarify atomics compatibility between C and C++ [N 2741]

Bhakta: C does have recommended practice, just no recommended practice in common with C++. Perhaps recommended practice should be outside of C (or C++) standard. It make no sense to reference C++ since we do not do that anywhere else in the standard.
Myers: We do have one thing in the draft saying "a C implementation". We used to have more things about C++. There is also the ABI issues of ABI's not updated due to atomics.
Gustedt: I am fine with the text as it is here. We could go further and state something like: This is an ABI issue, and platforms that implement it should be careful exposing these ABI's to other languages.
Sebor: If our standard does not mention C++, that is an omission. I am in favor of the text as is.
Gilding: Things like alignment & size differences make me wonder: Are there in-the-wild implementations using locks? At one point there were none that use locks, or else the locks were nowhere near the object in memory.
Boehm: The main discrepancy between implementations have nothing to do with how locks are handled. Usually alignment differences come from trade-off between underlying type and making lock-free implementations possible. On 32-bit platforms, double is 4-byte aligned, then what is _Atomic(double)? Being 8-byte-aligned allows lock-free implementations.
Voutilainen: To what extent has this been implemented? There were attempts to derive from atomic types. But not possible when an the atomic type is not represented as a struct type.
Boehm: The underlying mechanisms in the different languages can be different. The atomic in this proposal is an alias for the C++ type. There are ABI compatibility issues, but implementors are on board with fixing that.
Keaton: We are out of time. Boehm, can we discuss this paper in a future meeting?
Boehm: I do not have good direction at the moment as to what to change. The text for question #2 should be changed to be less C++-specific in places.
Keaton: Perhaps you should discuss it on the mailing list?
Boehm: OK.

- 5.22 Seacord, C Identifier Syntax using Unicode Standard Annex 31 [N 2777]

Myers: Modify s6.4.2.1p2. First, this is a constraint on universal character names (UCN's) that we should keep as a constraint, Second, in n2596 (C draft standard) this is in section s6.4.3p2.
Seacord: This is central to the paper…it lets us defer to UAX for what a universal character name is.
Bhakta: I like the general idea of this. Regarding 0-width joiners not being allowed: I do not believe that is true universally.
Downey: Yes, but there are other ways of freezing that.
Opinion Poll: Does the committee favor something along the lines N2777 to be added to C23? 18-1-3 Clear sentiment to proceed
Wiedijk: There are other implementation-defined chars in the syntax for identifiers. If my platform gains users by allowing emojis, is that possible or forbidden?
Bhakta: The workarounds you cite are workarounds. What is the technical reason for removing the non-joining space?
Downey: For other space characters, they are confused with whitespace. Which means the code you read does not do what you think it does.

- 5.23 Gustedt, Properly define blocks as part of the grammar [N 2739]

Myers: You do not need attributes to have things such as scope being a contiguous region; they can be dis-contiguous in function parameter definitions.
Gustedt: It is interesting input that scope is not contiguous.
Ballman: Is there a reason why labeled-statement is not using secondary-block and not statement as well?
Gustedt: That would define a scope for every labeled-statement which is not what we want.
Wiedijk: You mentioned lambdas? What happens to these changes if lambdas do not make it into C23?
Banham: What is the benefit case for an explicit grammatical term of "secondary-block"?

Gustedt: We are gaining a clearer direct syntax iteration. It becomes easier to teach the life of compound literals.
Banham: The secondary block does not implement any curly brackets.
Gustedt: It does not, but it can be another for loop, which has identifiers. It could be a block statement with identifiers. In C++ you can add declarations to the expression of an if or while statement.
Banham: Why the words "primary" and "secondary"?
Gustedt: A secondary-block is always inside a primary-block.
Gilding: Determining whether parameter lifespan change is static or automatic can be determined by portable code.
Gustedt: Yes.
Gilding: In Footnote 1, we discussed a syntax change allowing you to put the size of an array after the array itself.
Opinion Poll: Is a compound literal that occurs in the parameter list of a function definition associated with the block of the function? Yes=automatic variable, No=static. 13-0-8 Clear direction
Opinion Poll: Is WG14 in favor of adopting something along the lines of N2739 into C23? 12-1-8 Many abstentions but people would like to see this.

Friday, 3 September

- 5.24 Meneide, _Imaginary_I and _Complex_I Qualifiers, revision 0 [N 2726]

Bhakta: Why was this added in the first place?
Meneide: I did ask the reflector, but no one has responded.
Gilding: Are these allowed to be rvalues? Are they addressable?
Meneide: I assumed they are addressable rvalues.
Bhakta: An rvalue is not necessarily addressable. Lvalues are.
Seacord: In the draft standard, there are 13 occurrences where a macro expands a constant expression where that expression is not of type const. So if this is wrong, we could fix it & the others later.
Myers: We might want to add actual complex constants to the language, but only when we get proposals.
Bhakta: Is there anything that would break with this change, in an implementation that did have const float complex?
Meneide: Someone took the type of imaginary I and their typeof preserves on the const qualifier. That was the original motivation for this paper.
Wiedijk: In initializers you have this complex macro in 7.9.3, That gives you a complex constant back. So maybe this const qualifier is for something like that?
Gilding: Any code that depends on this being qualified does not know that it is doing so.
Decision Poll: Would WG14 like to adopt N2726 as is into C23? 19-0-2 So it goes into C23

- 5.25 Meneide, Consistent, Warningless, and Intuitive Initialization with {}, revision 0 [N 2727]

Myers: Some versions have levels of indirection. some duplicate rules for static storage. We ought to rearrange things like zero-initialization or deep auto-initialization, instead of saying "this is initialized like that". Also, separate zero-initialization and how an array size is determined.
Myers: Yes, that would be a second paper. I do need to figure out how to add tentative definitions.
Steenberg: This only zeroes things out if you have empty initializers, right?
Meneide: Right. GNU has an initializer that uses patterns; but that is not included in this paper.
Gustedt: First, you are using a strategy to initialize the largest member of a union. This would be a semantic change, and not unique. Second, the paper says nothing about compound literals. Next, this feature allows initializing any data type for which we know its type. This would be nice in the future for things like mutexes. Also I like 4.7, add initializers to VLA's, at least by default.
Meneide: Uecker suggested the VLA change. If you look at mbstate_t, it says if you get the address of an mbstate_t objects, you can use { } to initialize it without knowing its type. There are subtle differences between memset(…,0) and zero-initialization, on types where a 0 value is not all-bits-0. Using the largest member of a union rather than the first member follows existing practice in Clang & other compilers. We need to figure out how to resolve ambiguities (between two maximal union elements).
Bhakta: It would be better to initialize the first member of a union. But I am torn because the goal of the standard is to standardize existing practice. Also, you did allow the brace initialization to be used for scalars.
Meneide: That was intentional. For floating-point types, "= { }" differs from "= {0}".
Krause: I am uncomfortable with allowing this to initialize things like atomic_flags, which have their own initialization. Also, initializers have a messy syntax.
Meneide: People would be upset if we took out / deprecated some currently-working features. I prefer to fix this because it is a compatibility issue with C++. With regard to opaque types, I do not think there is a way to prevent brace-initialization for atomic types. The empty-brace initialization is an old foot-gun, not introduced by this paper.
Jabot: First, current practice is different from C++ and confusing. Also, C++ uses the first member for the union, so using the largest type for union is incompatible with C++.
Meneide: C++ is not a problem, because if you try to read from a union not properly written to, that is undefined behavior. But not in C. For a union(float, double), you can write to the float & read from the double, but the value is unspecified. Perhaps, for unions, the first member is initialized, but the rest of the union are all statically-initialized to 0, or something similar.

Gilding: It would be unrealistic to remove brace elision. It would annoy people who then have to create gigantic initializers. Also, allowing the largest union member is confusing.
Myers: Also with regard to obsolescing brace elision, I would worry about anonymous structs.
Steenberg: If there is ambiguity, I would rather have it not work at all and produce an error. There are ways in C to explicitly specify what you are initializing.
Krause: Initializing padding bits to 0 is done by C++…is there other motivation for this proposal to zero out padding bits?
Meneide: C has the same rule for zero-initializing padding bits outside of a union. The C draft standard s6.7.9 specifies padding bits are initialized to 0.
Opinion Poll: Would WG14 like to adopt something along the lines of N2727 into C23? 19-1-2 So clear direction
Bhakta: I voted no although I'm in favor in general. I want that union case to be the first member. I also want more clarity.
Gilding: Is there reason not to lift it for a single item?
Opinion Poll: Would WG14 like something along the lines of lifting the restrictions from VLA's to be initialize-able by an empty initializer as specified in N2727? 15-0-6 So clear direction

- 5.26 Meneide, char16_t & char32_t string literals shall be UTF-16 & UTF-32, revision 0 [N 2728]

  Jabot: If char16_t & char32_t are even worse than wchar_t. If C does not adopt this proposal, that creates compatibility issue with C++.
  Gustedt: I like this proposal. One suggestion: It would be good to have fault encodings be constraint violations. If people manage to have escape sequences that compose invalid Unicode, this should be diagnosed at compile time.
  Meneide: We do address escape sequences. We do allow invalid UTF- 8, 16, 32, but only for numeric escape sequences. We found that in programs, people would use numeric escape sequences to insert valid encodings in their own platforms. So such "invalid" encodings must not be compile-time constraint violation (of C).
  Gustedt: That makes sense for UTF-8, but not UTF-16 or UTF-32.
  Meneide: I agree that any value that is larger than a 21-bit limit should be a constraint violation. As for surrogates, people do use them. To be consistent with C++ we would just allow it. I would prefer to keep the current direction. If you want well-formed UTF, you should use "\u" (or "\U") escape sequences.
  Jabot: I agree with Meneide. It is not existing practice, nor what C++ chose. In practice it is not much of an issue. If someone wants to argue for numeric escape sequences, we can do that.
  Krause: Are we opening a can of worms with talking about invalid UTF items? Things like emojis? Where do we stop?
  Gustedt: That was yesterday, when talking about identifiers.
  Myers: With regard to invalid escape sequences. I agree they should be allowed. Certainly one case where you might get odd UTF-16 is Windows file names. You could be getting them from a filesystem if it uses them. N2653 and this paper affect each other; we will need to accept whichever one comes last carefully.
  Seacord: This makes it clearer that the purpose of the u8 encoding-prefix is to represent 7-bit ASCII characters. Should we also change "UTF-8 character constant" and refer to this as an "ASCII character constant"?
  Meneide: Yes, but the wording should include "not just ASCII". It might be better left the way it is; I am not sure.
  Jabot: I would argue that for character type, you can specify invalid UTF-8 characters. It is useful to talk of these as code units, not characters.
  Bhakta: It is a good idea to take the paper as is. I prefer current wording due to consistency. If that is something we want changed, that would be good for a future paper.
  Decision Poll: Would WG14 like to adopt N2728 into C23 as is? 16-0-3 So it goes in
  Bhakta: I am curious about what corrections should arise in a future paper.
  Gustedt: I found a typo for "u8" (should be "U8").

- 5.27 Meneide, Transparent Function Aliases, revision 0 [N 2729]

  Voutilainen: It is not like a typedef; it aliases a function, not a type.
  Meneide: Right. It could have the syntax of a typedef.
  Myers: With different syntax, you ought to allow an attribute on the alias. It could be deprecated or maybe-unused.
  Meneide: Yes, I should work that into the grammar.
  Voutilainen: We had debates on this matter in WG21. There are various quirks involved here.
  Ballman: I agree with Voutilainen. But from a user perspective, users want to attach attributes to anything with a name. So I am that sympathetic to platform developers saying "This is hard".
  Voutilainen: I agree in the sense that we do have existing implementation practice. It is certainly not insurmountable.
  Opinion Poll: Would WG14 like something along the lines of N2729 into C2x? 8-3-11 Many abstentions, but in general wanting to see more.
  Banham: What is WG14's stance on namespaces, a la C++?

6. Clarification Requests

The previous queue of clarification requests has been processed.

7. Other Business

The following papers were deferred to future meetings

New proposals for C23

- 7.1 Thomas, C23 proposal - Annex F overflow and underflow [N 2747]
- 7.5 honermann, char8_t: a type for utf-8 characters and strings (revision 1) [n 2653]
- 7.6 johnson, length modifiers for unicode character and string types [n 2761]
- 7.7 krause, unsigned long and unsigned long long bit-fields [n 2774]
- 7.8 uecker, variably-modified types [n 2778]
- 7.9 wiedijk, types do not have types [n 2781]
- 7.10 Uecker, remove undefined behavior for incomplete types of function parameters [n 2770]
- 7.11 Uecker, c23 atomics, Issues and Proposed Solutions [N 2771]
- 7.12 Ballman, Literal suffixes for bit-precise integers [N 2775]
- 7.13 Seacord, Volatile C++ Compatibility [N 2743]
- 7.14 Múgica, Memory layout of union members [N 2788]

Continuing proposals and other papers

- 7.15 Gustedt, Add new optional time bases v4 [N 2647]
- 7.16 Gustedt, type inference for variable definitions and function returns v4 [N 2735]
- 7.17 Gustedt, Simplementatione lambdas v4 [N 2736]
- 7.18 Meneide, Not-So-Magic: typeof(), revision 3 [N 2724]
- 7.19 Gustedt, Improve type generic programming v3 [N 2734]
- 7.20 Krause, @ and $ in source and execution character set [N 2701]
- 7.21 Meneide, Preprocessor embed, revision 4 [N 2725]
- 7.22 Tydeman, DFP: Quantum exponent of NaN (version 2) [N 2754]
- 7.23 Ballman, Fixes for potentially reserved identifiers [N 2762]
- 7.24 Ballman, Adding a Fundamental Type for N-bit integers (updates N2709) [N 2763]
- -7.25 Ballman, The noreturn attribute (updates N2700) [N 2764].- (Correction: adopted, clarification for the editors)
- 7.26 Steenberg, Redefining undefined Behavior [N 2769]
- 7.27 Uecker, Consistency of Parameters Declared as Arrays [N 2779]
- 7.28 Uecker, Forward Declaration of Parameters [N 2780]

Revisions:

- 7.29 (FKA 3.4) A revised charter for an undefined Behavior Study Group

  Decision Poll: Should WG14 create an undefined behavior SG with this charter? 12-3-5 The SG is created with this charter.

- 7.30 (FKA 8.1) Decide whether to add an extra day to the next meeting & face-to-vace meeting in January.

  Keaton: Herb Sutter and I are not willing to convene a face-to-face meeting unless we could restrict it to vaccinated people.
  Wiedijk: Some vaccines work better than others.
  Keaton: We do not have the equipment or knowledge to differentiate or know how to deal with someone who is unable to be vaccinated. A vaccine mandate that excludes exceptions is inherently unfair.
  Gustedt: Currently the US does not recognize all vaccines recognized in Europe. Maybe another host country would be worthwhile.
  Keaton: I will pull that up if we get that far. The host has paid approximately $75,000. After Oct second, they will have another commitment over $100,000 (for WG14 and WG21).
  Seacord: Vaccinated people can also get COVID. You could require a recent test.
  Keaton: We had not intended a test mandate in order to keep things simple.
  Svoboda: If COVID gets worse, do we get overridden, or denied permission? Also, most vaccine mandates allow for exceptions. And what about a mask mandate?
  Keaton: If the pandemic gets worse, we should agree to shut down the meeting ourselves (unless ISO or someone else does this for us). So vote on the notion that COVID does not get worse.
  Ballman: As the host, we are fine if WG14 chooses not to meet face-to-face. We would prefer a "no" vote now rather than after October.
  Keaton: Thank you.

Krause: Health experts do predict things will get worse in January. Travel may not be possible. Quality of people dialing-in to face-to-face meetings was always bad. I think it would be good to not plan for face-to-face meeting in January.
Gustedt: I am also skeptical of the quality of a mixed meeting. Our current (zoom) quality is much better now.
Seacord: This is not a vote about what happens in January; it is about what direction to give Intel, our host. You can always attend a face-to-face meeting virtually.
Keaton: If we have a hybrid meeting, we would ask everyone in the room on zoom, with present people having their audio turned off. That does not address audio problems, but makes things a little smoother.
Myers: I am skeptical of an in-person meeting in January. We should consider a second meeting perhaps earlier than the January meeting.
Svoboda: Has anyone had a good experience at a hybrid meeting?
Keaton: I see one check-mark & one X (on Zoom).
Yodaiken: I do not expect things will be better in January.
Wiedijk: I hear negative sentiment, but I miss this. Here in Amsterdam, we are out of the pandemic…everyone is mask-less. Might it be possible to get a very large room everyone can social-distance in?
Keaton: JTC-1 plenaries operate this way. Those are huge meeting rooms with space for approximately 350 people. But that is beyond our capabilities.
Ballman: Our facilities are larger rooms for WG21. This could work for WG14 but not WG21. The hotel threw in the Wg14 meeting room for free after we reserved rooms for WG21.
Krause: When does WG21 decide about this?
Keaton: In the next week or two.
Pygott: I am feeling dubious about traveling to Portland in January. I am looking at a hybrid or virtual meeting. If you are working normal hours in Portland (9am-5pm), that is 4pm-1am in the UK.
Keaton: Yes, we would have to adjust our hours. Which may mean fewer hours.
Banham: Booking the plane may mean you need to cancel a flight if the face-to-face meeting gets canceled. That provides Much benefit for the in-person meetings. Those benefits come from around everything that happens outside the meeting.
Seacord: Adding up uncertainties suggests the number of rooms WG14 occupies in Portland will be small. We will not be that influential on whether or not Intel puts this money down.
Gustedt: If we do a hybrid meeting, do we need two weeks?
Keaton: Yes. It would require modifying the schedule.
Wiedijk: The poll should be: "Do we want a face-to-face meeting?"
Keaton: It would not be what JTC1 calls a hybrid meeting. Which has a face-to-face component, but remote attendees have equivalent status.
Seacord: Poll: Would you attend in-person a hybrid meeting?
Keaton: There might be people who do not want one but would go if it happened.
Decision Poll: Would WG14 want a hybrid meeting in January rather than a virtual meeting? 4-14-3 Clear sentiment not to have a hybrid meeting.
Wiedijk: If WG21 decides to do it anyway, does that change anyone's opinion here?
Banham: Should we poll on when people might want a face-to-face meeting?
Gustedt: Wiedijk: I would not go.
Voutilainen: If WG21 chooses to have a meeting, that is pretty final for this year from my perspective. It is easier to cancel travel, but harder to resurrect a canceled meeting.
Keaton: I am going to recommend that we have a 5-day meeting rather than 6-day meeting because November is crowded, but does anyone want to advocate for a 6-day meeting?
Krause: Could we add the first day of Thanksgiving week for the meeting?
Decision Poll: Should WG14 add Monday November 2second as a sixth day to its November meeting? 7-4-10 With that many abstentions, we will not get enough value out of that extra day.
Seacord: Could we take a poll about eliminating floating-point day?
Keaton: I do not think so.

8. Resolutions, and Decisions reached

8.2 Review of Decisions Reached

Does WG14 wish to adopt the proposal N2686 into C23? 17-0-1 It goes into C23.
Would WG14 like to see N2688 Proposed Wording 1 adopted into C23? 15-0-5 So it goes into C23.
Would WG14 like to adopt N2710, with the change "LDL SNAN" to "LDBL SNAN", into C23? 16-0-2 So it goes into C23
Would WG14 like to adopt N2711 into C23? 11-0-6 So this goes into C23
Would WG14 like to adopt N2713 into C23? 10-2-7 This goes into C23
Would WG14 like to adopt N2714, with the change of "returns NaN" to "returns a NaN", into C23? 17-0-2 So it goes in
Would WG14 like to adopt N2715, with the addition of "potentially" back into the phrase (that is, it should not be removed), into C23? 15-0-5 So this goes in.
Would WG14 like to adopt the first and third change in N2716 into C23? 13-0-5 So that goes into C23

Would WG14 like to adopt the second change in N2716 into C23? 1-3-13 no consensus to make this change
Would WG14 like to adopt the change, but not the footnote, from N2745 into C23? 12-0-7 So it goes into C23.
Would WG14 like to adopt changes 2 and 3 from N2748 into C23? 16-0-2 So it goes into C23
Would WG14 like to adopt N2749 into C23? 11-0-6 So it goes into C23
Would WG14 like to adopt N2755, changing "implemenation" to "implementation", into C23? 14-1-3 So it goes into C23
Would WG14 like to adopt N2799 as is into C23? 23-0-1 So it goes into C23
Would WG14 like to adopt wording candidate 2 in N2801 to be adopted into C23? 17-0-5 So it goes into C23
Would WG14 like to adopt N2726 as is into C23? 19-0-2 So it goes into C23
Would WG14 like to adopt N2728 into C23 as is? 16-0-3 So it goes in
Should WG14 create an undefined behavior SG with this charter? 12-3-5 The SG is created with this charter.
Would WG14 want a hybrid meeting in January rather than a virtual meeting? 4-14-3 Clear sentiment not to have a hybrid meeting.
Should WG14 add Monday November 22nd as a sixth day to its November meeting? 7-4-10 With that many abstentions, we will not get enough value out of that extra day.

8.3 Review of Action Items

Keaton: Notify the author of N2684 of WG14's poll.
Seacord: Submit a paper standardizing that if the arguments to calloc() wrap when multiplied, calloc() shall return NULL.
Svoboda: Write a proposal to clean up C standard on wording on integer overflow (especially for unsigned integers)

9. PL22.11 Business

10. Thanks to Host

0.1 Thanks and apologies to Alex Gilding, the originally intended host

10.2 Thanks to ISO for supplying Zoom capabilities

11. Adjournment (PL22.11 motion)

Moved by Seacord, seconded by Pygott. No objections. Meeting adjourned Friday, September 3, 2021 at 16:00 UTC.