Final Minutes for 14 - 18 June, 2021

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

WG 14 / N 2802 (fka N 2768)

Dates and Times

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

14 June, 2021	13:30 – 17:00 UTC
15 June, 2021	13:30 – 17:00 UTC
16 June, 2021	13:30 – 17:00 UTC
17 June, 2021	13:30 – 17:00 UTC
18 June, 2021	13:30 – 17:00 UTC

Meeting Location

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 convenor 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
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	
Melanie Blower	Intel	USA	
Alex Gilding	Perforce / Programming Research Ltd.	USA	
David Goldblatt	Facebook	USA	
Jens Gustedt	INRIA	France	France NB
Barry Hedquist	Perennial	USA	PL22.11 IR

Name	Organization	NB	Notes
Tommy Hoffner	Intel	USA	
David Keaton	Keaton Consulting	USA	Convener
Will Klieber	CERT/SEI/CMU	USA	
Philipp Krause	Albert-Ludwigs-Universität Freiburg	Germany	
JeanHeyd Meneide	NEN	Netherlands	Netherlands NB
Maged Michel	Facebook	USA	
Joseph Myers	CodeSourcery / Siemens	UK	
Miguel Ojeda	UNE	Spain	Spain NB
John Plaice	Grammatech	USA	
Clive Pygott	LDRA Inc.	USA	WG23 Liaison
Mark Santaniello	Facebook	USA	
Robert Seacord	NCC Group	USA	
Peter Sewell	University of Cambridge	UK	Memory Model SG Chair
Eskil Steenberg	Quel Solaar	Sweden	Invited Guest
Nick Stoughton	The Austin Group, ISO/IEC JTC 1	USA	Austin Group Liaison
David Svoboda	CERT/SEI/CMU	USA	Scribe
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	

1.3 Procedures for this Meeting (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 2690] (WG 14 motion)

Motion to approve: Tydeman. Seconded: Stoughton.
Tydeman: I emailed typos to Svoboda
Svoboda: I have received them, thanks!
Keaton: Any objections? (none). Approved.

1.6 Review of Action Items and Resolutions

Action Item: *Tydeman*: Will bring this issue (POW(1, -Infinity)) to IEEE 754 and get their opinion. [N 2642]
Tydeman: Done. Sent email, zero response.
Action Item: *Tydeman*: Will search for other uses of the term "negative" in the draft standard. [N 2643]
Tydeman: Done. Found nothing else that needed changes.

1.7 Approval of Agenda [N 2758] (PL22.11 motion, WG 14 motion)

Motion to approve (for both PL22.11 and WG14): Tydeman. Seconded: Hedquist
Krause: Why are there two features for Gustedt's lambda?
Keaton: The provenance-aware memory model will need half the time it was allocated. One is Uecker's related paper. We have a large backlog. I will propose the possibility of extending this meeting to Saturday or Monday.
Ballman: Please do not move bit-precise integer type (5.7) from Wednesday to Tuesday because Hoffner cannot attend Tuesday.
Krause: Saturday, Sunday, and Monday are fine for me, but not Tuesday.
Hedquist: Monday works better for me.
Bhakta: It would be better if we had known this before the mailing deadline. It is not a good idea to modify agenda during the meeting without full consensus. I am hesitant to extend the meeting.
Gustedt: We should first address all papers that we have not yet seen.
Krause: Could we schedule a 1-week meeting between now and October?
Hedquist: We need to get through the .11 items by the end of Friday.
Svoboda: We usually end early.
Stoughton: Could we start 30 minutes earlier and end 30 minutes later?
Keaton: It is difficult to keep people engaged for more than 3 hours a day. I am withdrawing my proposal to add a day to this meeting.
Keaton: Any objections to agenda? (none). Approved.
Keaton: Any objections to filling out section 7 items if we have time? (none). Approved.

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

Krause: Is the reflector broken?
Pygott: I have not received copies back from the reflector
Svoboda: I concur, it has been sporadic last week.
Keaton: I will investigate. (done during meeting)
Action Item: *Keaton*: Investigate problems with the WG14 reflector. (done during meeting)

2. Reports on Liaison Activities

2.1 ISO, IEC, JTC 1, SC 22

2.2 PL22.11/WG 14

Keaton: Do we want to decide when the next revision (after C23) of the C standard should be published? We do not need to initiate a new project right away.
Gilding: Are there any advantages to an immediate revision vs. doing it later?
Keaton: Another question: Should the next revision be a bugfix release or should it contain new features? The advantage is that those who miss C23 know the next revision is happening soon. The disadvantage is that this means more work for us.
Seacord: I like starting a new revision immediately, and not limiting it to bugfixes. There are lots of documents that will miss the C23 boat.
Gustedt: I agree with Seacord. The backlog shows we would have gotten some features earlier. For example lambda would have arrived in C earlier than C23.
Ballman: Do implementors have any input from their users? C++ users felt that a 3-year C++ release was not "stabilized". More frequent releases seems to mean more risk for users.
Myers: We must pay more attention to not using invention if we use a faster release cycle.
Seacord: I like the flexibility that adding new features affords us. It seems unnecessary to handicap ourselves. C++ is a really different language (e.g. move fast and break stuff). C is not equivalent in that we do not break existing code.
Bhakta: Our community expects well-thought-out features. Also many new features come into compilers ahead of us. So we are better with a slower release schedule. My users think that even C++11 has not been quickly adopted, and likewise for C11. We may lose users

with a faster release schedule.
Ballman: It is restrictive to limit ourselves to bugfixes. When C++ switched to a 3-year model matches when their standard was considered unstable by my users. I do not think anyone is proposing changing the rules on inventiveness. I have had users suggest they prefer that C stop adopting new features.
Gilding: C11 is restricted to those who follow MISRA. My users pick a safety standard and use that to determine which version of C to use.
Bhakta: We are seeing other languages have feature XYZ so we should have it too. A lot of papers we should not be approving until we see C implementations of them.
Seacord: I hear two camps: Let us put C in a nursing home vs. let us try to keep C a vital language into the future. The perception out there is that C is past its usefulness. Showing that you are trying to modernize the language is useful for the latter camp.
Bhakta: That mischaracterizes the language. "Vital evolving C" is really C++.
Meneides: We have many things we could add that people want. We can keep very busy without being inventive.
Gustedt: I disagree that many proposed features like lambdas and nested functions are not implemented by compilers.
Action Item: *Keaton*: Put standardization schedule as an agenda item on the full January agenda.

2.3 PL22.16/WG 21

Wong: Our most recent meeting was last week; we are still working on C++23. We are starting a new TS on Concurrency. We plan to be virtual for at least the next meeting. Our February meeting might be non-virtual.

2.4 PL22

Keaton: The next PL22 meeting is happening this Thursday after the WG14 meeting.

2.5 WG 23

Pygott: We are unable to get ISO to let us make various parts of WG23 free. We have a set of TR's rather than standards. We wanted to move language-independent and some language-specific parts into their own standards.
Keaton: ISO made changes to directives to not directly affect WG14, but they hurt WG23 badly. WG23 has many TR's for avoiding vulnerabilities. ISO now forbids new TR's to be free. TR's may also not provide advice, so WG23's TR's may not be TR's anymore, hence WG23 is converting them to standards.

2.6 MISRA C

Pygott: We are still meeting and working on the next version.
Bagnara: We are still incorporating features from C11. There is worry about us to "volatile" what the C++ working group is doing to "volatile".

2.7 Austin Group

The Austin Group Liaison Statement [N 2723]
Stoughton:

The Austin group is a collaboration between SC22, IEEE, and The Open Group to produce the POSIX standard, which includes all of the C library interfaces. We defer to the C standard when any question arises. We are upgrading our standard to be based on C17 rather than C99. It is unlikely that this revision will be able to support C23.
There are two bugs we have had for a while:
The first is Bug 700, where we must clarify strtoul()'s behavior on negative numbers when converting to an unsigned value. Both POSIX and C have the same kind of ambiguity, but every implementation behaves the same way. Are there any comments on that question? (none) If we adjust wording to match the implementations, I can submit that for the next meeting. I would expect no strong push-back.
Second item: What happens with whether remquo result should be a NaN? Does anyone here have an opinion?

Tydeman: I have not studied this yet but will respond.
Action Item: *Tydeman*: will repsond to Austin Group about whether the remquo result should be a NaN.
Voutilainen: From WG21 perspective, it is not realistic to expect C++ to be based on anything other than previous C standard. WG21 has a feature freeze now.
Bhakta: I apologize to Steenberg, I muted you when I heard background noise. Should Keaton or I mute people when we hear noise and see non-speaking people off mute?
Svoboda: Yes, please mute non-speakers if background noiose appears.

2.8 Other Liaison Activities

3. Reports from Study Groups

3.1 C Floating Point activity report

Tydeman: We are having monthly meetings.
Bhakta: We have a proposal on what to do with Part 5, which catches us up to 2021's IEC 60559 (and IEEE 754) standard. IEEE has not started a new standard yet. So from a C floating-point-of-view we will have caught up.

3.2 C Memory Object Model Study Group

Sewell: You will see everything tomorrow in our agenda item.

3.3 C and C++ Compatibility Study Group

Omnibus of WG21 Papers (Mar 2021) [N 2704]
Ballman:

There have been 3 meetings since our last WG14 meeting. We are making smooth process, but our backlog is getting bigger, which suggests that this group is useful. There is desire for people to find helpers on either committee. Olivier Giroux generously offereed 1 day of every face-to-face meeting where SG1 (in WG21) would review WG14 papers for compatibility. We reported C23 status to WG21 plenary. There is lots of happy sentiment on the time WG14 has spent on the C/C++ compatibility papers. Some people were surprised at the floating-point changes, and how they, especially the new types, impact C++.

Bhakta: It is good to hear about the floating-point reaction. The floating-point group would like C++ members to attend.
Myers: With regard to adding new types: Does C++ have a view for things like float64 and float32x?
Ballman: I have not heard anything specific.
Bhakta: When we hashed this out, CFP was not tied to the original decision on these types.
Ballman: It might be useful for CFP to target a paper to WG21 summarizing changes to C floating-point.
Action Item: *Bhakta* & *Ballman*: Let us have CFP SG target a paper to WG21 summarizing changes to C floating-point.
Voutilainen: It is good to take this to the liaison study group.
Ballman: The C++ Compatibility SG is also publishing omnibus papers, so they are in the WG21 document system. (No action item needed on WG14's part).

4. Future Meetings

4.1 Future Meeting Schedule

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

Keaton:

ISO has a new declaration: a ban on face-to-face meetings through September, and they may extend the ban another month. However, they allow committees to apply for an exception to the ban to test for hybrid face-to-face and remote meetings. We would need to apply for 4 exceptions rather than two, but we could meet face-to-face. The USA CDC is still banning people from many countries from entering the US, including national bodies that attend WG14.
With 33 weeks until January meeting. I propose 11 weeks to next meeting and 11 weeks until subsequent meeting.

Krause: With such a big backlog, should it be 3 meetings rather than two?
Keaton: I prefer adding a day to 2 meetings, but let us keep that in mind.
Myers: August 30 is a holiday in the UK.
Keaton: The following week has a US holiday. 8 weeks from now is August 9. 11 weeks is August 30.
Gustedt: I would have problems with August 9.
Bhakta: Do we have a requirement to go through all the papers (in our backlog)? Does that force us to have more meeting time?
Keaton: It is not a procedural requirement. I worked backwards from when ISO expects us to publish the standard.
Gustedt: I reiterate my suggestion that we prioritize papers we have not seen yet.
Keaton: I am not hearing issues with these two weeks (e.g. Saturday). Are there any objections to adding Saturday to these meetings?
Straw Poll: Do we wish to replace the October meeting with two meetings starting 30 August and 15 November? 15-0-2
Myers: Is there another day that people prefer?
Straw Poll: Should we add a sixth day to each of the meetings starting 30 August and 15 November? 10-4-4
Keaton: I would like to hear from anyone who objected?
Bhakta: I prefer separating work life and home life. I doubt I will get more than 5 days from my job.
Gustedt: It is difficult to schedule a sixth day.

Krause: We need a solution; we still have our backlog.
Keaton: Can we add Friday, August 27 to the first meeting, and decide then about the November meeting?
Ojeda: What about lengthening the day?
Keaton: Can people only handle 3 hour meetings?
Straw Poll: Should the committee add Friday, August 27 to the next meeting, making it a 6-day meeting? 9-1-8 I am thinking we should do this.
Krause: I think we should vote Monday vs. Saturday.
Straw Poll: Should the committee choose Saturday, September 4 vs. Friday, August 27 as part of the next meeting? 1-7-12 Saturday is not desired.
Straw Poll: Should the committee choose Monday, September 6 vs. Friday, August 27 as part of the next meeting? 3-7-8 So previous Friday wins.
Action Item: *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 occurs during the week of November 15.
Bhakta: How vehemently do people object to Friday August 27?
Keaton: The remaining non-virtual meetings are still scheduled:

- 31 January - 4 February, 2022 – Portland, Oregon, US (tentative)
- 11-15 July, 2022 – Strasbourg, France (tentative)

4.2 Future Mailing Deadlines

Ballman: Does this change mailing deadlines?
Keaton: Yes. I will include that in the document.
Ojeda: What happens to the papers in the next meetings?
Keaton: I will do my best to prioritize papers we have not seen before.
Gustedt: Strasbourg has once-a-month European parliament meetings and we do not want that to overlap with WG14.
Myers: Will you confirm what time (UTC) the meetings will be?
Keaton: Yes, adjusting for Daylight Savings Time.

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

- Pre-Virtual-202106 – 14 May 2021
- Post-Virutal-202106 – 9 July 2021
- Pre-Minneapolis – 3 September 2021
- Post-Minneapolis – 29 October 2021
- Pre-Portland – 31 December 2022
- Post-Portland – 25 February 2022
- Pre-Strasbourg – 10 June 2022
- Post-Strasbourg – 5 August 2022

5. Document Review

Monday

- 5.1 Working draft updates (if available)

  Meneides: I am making progress on Annex N. I am at section 12, (which is 3/4ths of the way through). Papers from last meeting will be merged soon, and I have the paper numbers. The actual new paper website is almost up and I have a paper number for it, I will be pushing that out soon too.
  Bhakta: What is Annex N?
  Keaton: That is Floating-Point Part 3.

- 5.2 Svoboda, Towards Integer Safety (updates N 2681) [N 2683] (wrap-up from previous meeting)

  Seacord: For the ckd_inexact() macro, the term "inexact" is a floating-point thing, not an integer thing. Does this have to indicate the failure condition? The macro would be better called "ckd_correct", or better yet "ckd_misrepresented".
  Svoboda: We used to use the term "overflow". But then I read 10967-1 which defines "overflow" and "inexact" as floating-point things. Also "overflow" as defined by 10967-1 probably does not apply to INT_MIN - 1. The term "exact" comes from Java's approach to safe integer operations.
  Gustedt: I am happy with core proposal. But the supplemental proposal defines new types, and we should better describe their properties. Can we use ckd_mk() in initializers? I want something along the lines of the supplemental? proposal, but not necessarily

the supplemental proposal as is. Why does the proposal speak of "infinite range"
Svoboda: The infinite-integer arithmetic wording came from GCC, and it was an improvement over the previous wording. This is addressed in the "Standard Arithmetic Conversions" section of N2683.
Krause: The intmax_t type is going out of the language soon.
Svoboda: Perhaps, but it is still part of the language today, so this proposal leaves it in.
Seacord: The core is ready, but the supplemental is far from ready. I have many questions about it: In a checked type, is the "inexact" flag changed by an operation? Is the value stored in x correctly represented? In the "ckd_value macro" section, about the wording: "If inexact flag is clear" …does that mean false? And the language that says "reduced by modular arithmetic with silent wrap-around on overflow"…can we just say two's-complement representation? And in section 7.2.4.5: "checked_op macros", some of these return result in a pointer to result type, but the description does not say what happens if the pointer is null. Is this a no-op or undefined behavior, or indicate an error? That needs to be specified.
Svoboda: The inexact flag is never changed, it is initialized with the proper value. The checked types provide no mechanism to change values, the values are only initialized when a checked-type object is constructed.
Seacord: Does that allocate storage? MISRA forbids dynamic allocation.
Gustedt: Is there pre-existing implementation for supplemental part?
Svoboda: The core proposal matches the GCC builtins, only changing names and argument ordering. The supplemental types are not part of the GCC builtins.
Gustedt: No implementation experience on supplemental types.
Ballman: Regarding the future library directions. It is unclear if the "ckd_" names should be "potentially reserved identifiers" or just "reserved identifiers".
Svoboda: Yes, that distinction did not exist when I started the paper. But I am fine with that change.
Ballman: Regarding implementation experience: did the GCC folks have any problems with the changes you made to the core builtins?
Svoboda: I have not asked them.
Bachmann: I do not want the behavior on null pointers to be restated in this proposal, because that opens the risk of inconsistency.
Bhakta: The null pointer case is already handled by the wording "shall be a modifiable lvalue". That makes it analogous to constraint violations, like undefined behavior.
Keaton: Svoboda, I suggest that we vote on the core proposal, and you work with the commenters on the supplemental proposal.
Svoboda: Agreed.
Straw Poll: Does the committee wish to adopt the Core proposal from N2683 into C23? 13-0-5 Core proposal goes into C23.
Ballman: This includes my proposed changes to future library directions, right? Please make sure an updated document goes in with that.
Straw Poll: Does the committee want the "ckd_" identifiers from N2683's Core proposal in future library directions to be potentially reserved identifiers? 18-0-0
Action Item: *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.

- 5.3 Gustedt, Type-generic lambdas v3 [N 2738]

  Keaton: We are out of time; we can discuss tomorrow. Gustedt, can you publish these slides somewhere?
  Gustedt: I will put links to these slides in tomorrow's chat.

Tuesday

- 5.5 A Provenance-aware Memory Object Model for C (3 hours)

  Working Draft TS 6010 [N 2676]
  Myers: We should not have the address-taken rule. Memcpy is out-of-scope here
  Wiedijk: There exist many calls that initialize some but not all local variables in a function, and I can memcpy them all (some uninitialized).
  Keaton: The address-taken exception was a concession to allow Not-A-Thing (NaT) to be accepted into the standard. If you can find a more clean way for the concession, that would be great.
  Uecker: It should be the same (without the address-taken exception). I do not think people understand the consequences of removing the address-taken exception.
  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
  Wiedijk: Would any naysayers please comment?
  Bhakta: Any code depending on current text would be broken.
  Banham: I am not clear why you would want the semantics to be different.
  Sewell: That means you should vote yes.
  Banham: Agreed.
  Wiedijk: My "no" vote would be a "yes" if this were more defined.

Keaton: I would have voted no but I presume you would find some other way to allow NaT.
Sewell: Adding NaT adds more undefined behavior.
Svoboda: The optimizer is assuming that undefined behavior cannot happen, which makes it jump to conclusions in your code example.
Steenberg: I think undefined behavior happens, but compilers use that to do surprising optimizations. That is a problem of undefined behavior as a concept.
Myers: The problem is not so much compilers saying this cannot happen, but an optimizer that simplifies code to prevent undefined behavior from happening.
Yodaiken: The compiler does not have to optimize, it could instead flag an error.
Ojeda: I would be fine with adding this kind of undefined behavior to C if it was otherwise safe.
Sewell: Clearly there are not entirely compatible views of undefined behavior here. At an uninitialized read, the implementation could give a compiler error or a non-silent runtime error, which is more restrictive than undefined behavior.
Steenberg: I would like the actual value to be indeterminate…0 is bad because it is not unexpected. I want a compiler to throw an error if it detects uninitialized reads.
Wiedijk: Zero is not an option…it will let buggy programs appear to work by getting 0. A program that uses this on a compiler that does not support it will break. This is a hodgepodge of different possibilities.
Bhakta: The argument always breaks down to this: Our charter says trust the programmer. Ballman adds "except for safety reasons". I say that part does not apply. I am arguing against option 0 (returning a 0 to uninitialized reads).
Gustedt: Do we want access to uninitialized reads to be a programming error? Assuming not, then I would prefer returning 0. If we consider uninitialized reads to be an error, we have to put some dead store value there.
Bagnara: I prefer option 0, where the compiler vendors provide a zero-initialization option, which hopefully gets optimized away for initialized variables. The current C standard does not prevent automatic initialization. So why should the standard force implementors to do anything?.
Sewell: What we have currently does not make people happy.
Ballman: Part of the charter is to standardize existing practice. I know four implementations that allow automatic initialization. Users do seem interested in having these options.
Ojeda: This behavior was updated in C11. I have a pending paper on uninitialized reads. Blindly trusting people to not make mistakes is bad strategy. I do not see the point of leaving variables uninitialized. What is the performance cost of pre-initializing variables?
Krause: LLVM is one of the most advanced compilers. For embedded compilers, cost would be much higher. If we change, then performance decreases by 4.5%. So that hurts speed and boosts energy consumption.
Wiedijk: I would like a compiler that makes the most trivial compile. If the compiler must either initialize or signal error on uninitialized reads, that is a strong reason to avoid the 0 option.
Sewell: You already have to do that for globals, because they are default-initialized.
Wiedijk: But that is not for locals.
Bhakta: Many customers do care about performance, a 5% performance difference matters. Every compiler vendor I know of has this option. I know 6 compilers where the usage of automatic initialization is very low. So the argument that we should support this because it is used is weak. We standardize only a few optimization flags, such as floating-point pragmas.
Sewell: I think you are arguing against just option 0, right?
Bhakta: I am arguing against all of these options.
Ojeda: If you migrate from old to new compilers, the new compiler provides flags to mimic old behavior. I do not see why the 5% argument is reply important. People who care about their initialized reads will keep the old behavior.
Steenberg: If you have uninitialized reads, the best thing is that the compiler notifies you during development, so as to minimize runtime impact.
Bachmann: As a user, an uninitialized read is a programming error. Any kind of automatic initialization is not useful at all.
Seacord: We have had this problem with uninitialized reads since C17, and we promised to fix it in the next release of the standard. The status quo is not an option. The standard is completely unclear on this issue. The easiest solution is to make it undefined behavior. Alternately, we could make it implementation-defined behavior, and enumerate existing behaviors, and eliminate undefined behavior.
Voutilainen: It is unclear to me that users do not want undefined behavior here. This has compatibility impact to C++ Liaison SG. Ballman, will you present it there?
Ballman: Yes. I would invite Sewell to present there, and try to add WG14 people there.
Keaton: After our first vote, I am concerned about the lack of foundation about functionality. I thought Sewell intended to find another way to allow NaT. Bhakta got the opposite impression. I am in favor of standardizing our functionality in a cleaner way, but not in favor of throwing out our current standard because I will want functionality back. We need more reality on what we vote on because otherwise we do not know what we would be giving up.
Sewell: This is different than a normal paper discussion. It is not clear what the appropriate options should be. I suggest we end with two straw polls.
Svoboda: One solution is to either eliminate trap representations for various types (besides unsigned char) or allow platforms to do so. That reduces undefined behavior to "reading an indeterminate value".
Sewell: But that is not something we are discussing today.
Voutilainen: With regard to defining "alternative options": How do we do that? do we make it a normative requirement, or leave it

open-ended?
Sewell: My feeling is that the alliterative options should be a small set to avoid confusion. I am unclear if it should include undefined behavior.
Voutilainen: Whether it includes undefined behavior would affect my vote. The question should be more specific.
Bhakta: For "alternative options", do you mean things like pragmas, or dialects of C?
Sewell: I have no opinion on what they should be yet.
Bhakta: Something like freestanding vs. hosted would be a big issue for me.
Sewell: What is the difference for you?
Bhakta: You can have compilers that support only freestanding vs. only hosted. I would want a compiler that supports all alternatives.
Myers: Maybe a straw poll on the question: What does the committee thinks about adding more dialects?
Gilding: A user can get a configuration of options with no pragmas or warnings and think their program is well-defined. That is what worries me about multiple defined options.
Keaton: The "stronger than undefined behavior" behavior; what does it mean?
Sewell: The semantics would be stronger than undefined behavior, because undefined behavior guarantees nothing.
Straw Poll: Does the committee support, in principle, the approach of defining a small number of alternative options for uninitialized-read semantics, allowing users to choose as they wish? 6-9-6 no sentiment to pursue
Straw Poll: Does the committee support, in principle, the approach of explicitly permitting, at the implementation's per-instance choice, either a compile-time error, a non-silent runtime error, or some other more-defined-than-undefined behavior for uninitialized-read semantics. 10-9-3 not really sufficient
Sewell: This next poll would make NaTs always be undefined behavior.
Straw Poll: Does the committee want to make uninitialized reads of scalar non-character typed objects of automatic storage duration always be undefined behavior? 10-9-3 not enough direction to say people are definitely in favor
Bhakta: For me, the unclear aspects affect a lot of the votes. I had to vote conservatively. This last poll lets us keep the NaT exception.
Sewell: The NaT exception rules everything in this approach.
Bhakta: We may have opposing viewpoints, or we might get better results with different poll questions.
Voutilainen: It seems premature to me to vote in favor of "always undefined behavior".

- 5.3 Gustedt, Type-generic lambdas v3 [N 2738], cont

  Gilding: I was worried that this would go into a pseudo-template direction. I am relieved that is not happening. Even though in section 2.1, you do not want lookahead or pre-compilation, we still do not know what the operations are and we cannot type-check them although we can assign it to a function pointer, or call after it. This would break a 1-pass compilation model. It would need a C++ model of storing the lambda text and parsing after type-checking its use.
  Gustedt: You only have to look ahead for function parameter declaration. Inside a lambda body, all types are clear. The first scan scans the body of the lambda for balanced expressions and types of the arguments. Then resolve types. It could be plugged into some preprocessor annotation, but I do not think that is necessary.
  Myers: Regarding Q1: "want to get for C23" is very different from "want to get for some later point". It is an impressive feature for C26 or 29, but I am skeptical of adding any lambda things for C23 until we have more implementation experience.
  Gustedt: If we do not add this to C23, we will see lots of ugly home-made code captures.
  Uecker: It is relatively easy to shuffle tokens as Gilding specified but it can be expensive. We lose simple compilers if we pass this. It seems completely unnecessary for the fundamental feature.
  Gustedt: What I had as first two examples does this token shuffling. I was striving for C++ compatibility. In order to be compatible, we have little choice on syntax.
  Uecker: C++ has a more complicated parser anyway.
  Svoboda: How compatible are these with C++ lambdas?
  Gustedt: Two issues with compatibility: We do not have trailing return types as C++ does. Also, C++ has the "mutable" keyword, which can go after the parameter list, which means all captures are not const-qualified. We could add those if requested.
  Ballman: I am glad to see this. In the paper type-generic lambdas are an incomplete type. Can I assign it to an automatic variable?
  Gustedt: C does not have templates. You cannot assign a type-generic lambda to an automatic variable; you have to complete type specification.
  Ballman: You can assign regular lambda to an auto variable, but not type-generic lambdas, right?
  Gustedt: Right. We could add that at a later stage.
  Krause: If this gets voted in, it is the first feature in C that cannot be handed by a lex/yacc parser?
  Gustedt: No, I have it supported by my lex/yacc parser.
  Straw Poll: Does the committee want type-generic lambdas along the lines of N2738? 10-6-5 general sentiment in favor, maybe we want lambdas but not along the lines of N2738.
  Gustedt: This proposal is about type-generic lambdas, not lambdas in general.

- 5.4 Gustedt, Lvalue closures v3 [N 2737]

Krause: Everything you propose with captures, I could just do with pointers.
Gustedt: I agree, unless your variable is a register.
Gilding: I like that references are not required, which is completely different from Clang blocks. The matrix implementation was very cool!
Bhakta: Regarding support for default capture mechanisms: If the "&" is not needed, why is that a value capture, not an lvalue capture? What is the point of a default?
Gustedt: For all identifiers you do not list there, you can use any automatic variable.
Myers: In the earlier slide example of accessing lambda from another lambda, why do you need lvalue captures?
Gustedt: It would be valid without the ampersand, but it would make a copy of the lambda, which gets complicated.

Wednesday

- 5.7 Ballman, Adding a Fundamental Type for N-bit integers (updates N2646) [N 2709]

  Myers: Why the annex approach? Is the annex intended to be conditionally normative?
  Ballman: No, it is normative.
  Myers: Then why would we want an annex?
  Ballman: Gustedt suggested an annex so we could specify semantics while leaving out implementation details. Personally I think the annex spec is harder to read. We do not want this to be a conditional feature because we like the portability.
  Myers: For the non-annex approach: What counts as a signed vs. unsigned type? The standard, bit-precise types and extended integer types are called signed integer types. "signed integer types" is repeated three times. This approach should use the same wording as annex approach.
  Ballman: Yes, you are right.
  Krause: None of the types shall be defined for bit-precise integer types. The current wording does not forbid any other typedefs to be bit-precise. It is not allowed to have uint24_t as a bit-precise int. I do not see the need to forbid types that are not standard integer types. But this could be changed later once the base proposal is in.
  Ballman: Yes, it is interesting that we do not talk about size_t or other such types. We did not want to surprise people and give them a bit-precise type rather than a "normal" integer type. Short and char types promote to int, and precise-bit types do not, that is why we prefer using standard types over bit-precise types.
  Bhakta: I am concerned with not making this conditionally normative. I do not think making this portable applies. That argument could apply to all conditionally-normative annexes. Is there a better argument against making this conditionally normative?
  Myers: We have "conditionally normative" features in the main standard too (outside of annexes). Decimal-floating-point, complexs are conditionally normative but in the main standard.
  Ballman: Because every platform supports integer types, they could support a bit-int up to their largest type, so there is no gain to users from making this conditionally normative.
  Bhakta: I disagree. I see portable code with #ifdef's on supporting atomics, VLAs, etc.
  Ballman: OK but what value do users get from "conditionally normative"? You have argued the value only for implementors so far. You can fake VLA's with other techniques, but you cannot fake bit-precise integers. I assume that we want everything to be portable, unless there exists a reason to make it non-portable.
  Bhakta: I disagree, but I understand your assumption.
  Yodaiken: How do these bit-precise integers work with the aliasing rules.
  Ballman: I am not aware of bit-precise integers behaving differently than other integers with regard to aliasing.
  Yodaiken: If you had a 256-bit type, can you look at it in 64-bit chunks?
  Ballman: I would memcpy() that into a buffer. I see Banham asked if there are any endian issues in this proposal. There are no more than usual.
  Myers: Endianness is part of the ABI considerations. If this gets accepted, we will see ABI proposals with various ABI groups.
  Ballman: Correct.
  Bhakta: Alternative 1 is the non-annex wording, right?
  Ballman: Right!
  Straw Poll: Does WG14 wish to adopt N2709, proposed wording alternative one, into C2x? 14-2-5 This goes into C23.
  Ballman: Since there were some wording, we should resolve it in a new document, and strip out alternative 2. We can also produce an additional document for Krause's point for typedefs in the standard library, as follow-up work.

- 5.8 Thomas, C2X proposal - signbit cleanup [N 2650]

  Straw Poll: Does WG14 wish to adopt N2650 into C23? 18-0-2 This goes into C23.

- 5.9 Thomas, C2X proposal - fabs and copysign cleanup [N 2651]

  Myers: These requirements may not be effective when they are defined to be a convert-format operation, that may lose information about the object.
  Svoboda: I thought math functions that apply to specific types did specify the type in the normative text; e.g. the fabs(x) function.

NaN is not a floating-point number but it is an element of that type.
Bhakta: We did not go that route, because you do not see that in other function types.
Hoffner: You are opening this up to strings.
Bhakta: We do not intend to open this up to strings.
Straw Poll: Does WG14 wish to adopt the change to 7.12.7.3 listed in N2651 into C23? 20-0-3 so it goes in.
Straw Poll: Does WG14 wish to adopt the change to 7.12.11.1 listed in N2651 into C23? 17-0-6 so it goes into C23.

- 5.10 Thomas, TS 18661-5 revision [N 2652]

  Myers: Is this about part 5? Have we agreed on part 4b or will that be proposed separately?
  Bhakta: Separately. There are proposals to bring it into C2x.
  Seacord: We discussed starting a new version of the C standard immediately, but here you are addressing floating-point features that might not make it into C23…do you want an immediate revision of the next standard?
  Bhakta: I have not expressed my views on whether we **should** do a new version. CFP is not committed to getting these revisions into C23.
  Seacord: Do you have a preference for when the next standard begins, based on getting these parts included?
  Bhakta: It makes sense to get these into C23, in order to concur with IEEE 2019 standard.
  Myers: Section 4b has less complexity than section 5.
  Krause: Will section 4b be incorporated into the new section 5?
  Bhakta: Section 4b should not be put into section 5 unless WG14 decides they do not want section 4b into the main standard.
  Straw Poll: Does WG14 wish for CFP to create a new revision of TS 18661-5 based on C23? 19-0-2 there will be a revision of Part 5.
  Action Item: *Keaton*: See what actions we need to take with ISO in order to start the TS 18661-5 revision.
  Keaton:

  I have done this…here is what I found:
  When we start a new project, we need a "new work item" proposal, which requires 5 national bodies.
  To revise a standard, we need SC22's permission.
  To revise a TS, the TS needs to be within its initial 6-year lifespan. TS 1861-5 is within its lifespan.

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

- 5.11 Thomas, C23 proposal - zeros compare equal [N 2670]

  Svoboda: If we are supporting two's-complement, are negative zero obsolete? Or is it only for floating-point negative zeros?
  Bhakta: It was not originally intended to be only for floating-point arithmetic, but I guess it is now.
  Seacord: For some scientific cases is there some way of distinguishing positive from negative 0?
  Bhakta: Well, you can examine the bits directly; there are mechanisms to do that. But they require knowing the bit-level representation of your 0. There is no way that is portable (outside of the signbit macro).
  Ballman: Section 6.5.8 already talks about floating-point specifically. It might be nice to say "positive floating-point zeros compare equal to negative floating-point zeros" for clarity.
  Bhakta: Noted. There are other types besides floating-point and integer types…some platforms have fixed-point types, for example. A future paper might suggest it.
  Straw Poll: Does WG14 wish to accept N2670 into C23? 19-1-3 so it goes into C23.
  Wiedijk: I am curious why you voted against this?
  Bhakta: For the reason Ballman mentioned: performance and requirements on other types (like fixed point). I would have voted in favor if it said "only floating-point types".
  Gilding: Other types still mean comparisons are not bit-wise; they focus on mathematical values.
  Wiedijk: I would like to hear another straw poll on whether floating-point should be added.
  Bhakta: As a CFP representative, I do not have words for that.
  Keaton: If someone wants to submit a paper for that, we can address it.

- 7.1 Thomas, C23 proposal - negative values [N 2671]

  Svoboda: Some questions: NANs are not numbers but they are floating-point objects, right? Is -Infinity < 0? Can complex numbers be negative?
  Bhakta: -Infinity < 0, but for complex numbers, it depends on who you ask. Personally, I think not.
  Svoboda: Will this change break anything in the complex number library?
  Bhakta: I do not think so.
  Myers: Have you reviewed uses of the word "negative" in the standard to make sure they do not need to be adjusted with this change? I see the printf "+" flag: "begins with a sign only when a negative is converted." That may need to be adjusted, and similarly for the wprintf() function as well.
  Tydeman: I have a paper with these changes, it is document N2643.

Bhakta: Yes, that paper is based on exactly your question.
Ballman: Could this break implementations of things like fsqrt()? (e.g. by causing a domain error)
Bhakta: This is not intended to change that.
Tydeman: The standard explicitly states that sqrt(-0) == -0.
Svoboda: Should not we vote on those papers together?
Keaton: We adopted N2643 in the previous meeting.
Bhakta: The two papers are mostly independent.
Seacord: -0 and NaNs are not negative values, but is the opposite of that true?
Bhakta: -0 is not a positive value, neither is (+) 0. A NaN is not a positive or negative value.
Seacord: This could be clearer, because negative and positive are not strict opposites of each other.
Straw Poll: Does WG14 wish to accept N2671 into C23? 16-0-2 So it goes into C23.
Bhakta: So we voted positive on the negative proposal.

- 7.2 Thomas, C23 proposal - 5.2.4.2.2 cleanup [N 2672]

  Straw Poll: Does WG14 wish to accept N2672 into C23? 21-0-1 So it goes into C23.

- 7.3 Uecker, Indeterminate Values and Trap Representations [N 2668]

  Svoboda: Is NaN a trap representation? (Does "value" mean "mathematical value" a la floating-point types?)
  Uecker: I think NaN is not a trap representation, because it represents a value of floating-point types.
  Myers: A signaling NaN can be a trap representation. But if an implementation does not handle signalling NaNs, it is not. The 80-bit x86 precision floating-point format acts a lot like signalling NaNs.
  Bhakta: NaN is a value, but it is not a number.
  Yodaiken: What is the reason for reading a trap undefined behavior, rather than implementation-defined behavior?
  Uecker: I do not know. We do not require the platform to check that whether a trap is a valid value. The text "implementation-defined" requires the implementation to define exactly what happens. That is not something we are proposing here.
  Gilding: I always assumed "indeterminate value" means "undetermined whether it is a value". If the behavior were implementation-defined, that would require special builds for things like valgrind. You can configure valgrind to break execution if an unspecified value is read.
  Yodaiken: Valgrind does not check for standards compliance…it could still say "you have an incorrect value in this boolean".
  Bhakta: You forbid signalling NaN from being traps. I do not believe that was the intent.
  Uecker: A signalling NaN is both a trap representation and a valid value.
  Bhakta: A signalling NaN "need not" represent a value. Reading the NaN produces the signal, and changes the value to a "quiet NaN" or "QNaN".
  Uecker: Reading a trap representation is undefined behavior. But you could specify the behavior like "raises a signal".
  Tydeman: Many words are now bold that were not bold in the standard.
  Uecker: I tried to highlight each verb to explain why we made a specific choice. It was not meant to make anything bold in the standard.
  Banham: Regarding Pointers not having a trap representation: Dereferencing invalid pointers can still trap.
  Uecker: The provenance TS will address dereferencing. We call end-of-life pointers "invalid", dereferencing them is undefined behavior but not trap representations.
  Banham: That suggests that some undefined behaviors can be traps.
  Ballman: Did the Memory Object Model SG review this paper?
  Uecker: This came out of discussions in that SG.
  Wiedijk: The word "valid"…what does it mean? It is not defined in Chapter 3. A past-lifetime pointer is invalid, which leads to indeterminate values which leads to valid value or trap representation. This is a maze of twisted definitions, all alike.
  Uecker: Solving all the open questions seems infeasible. "invalid pointer" already exists in the standard, but is used inconsistently. Pointers were distinct from objects, so we needed a different term (invalid) for dead pointers. A pointer value cannot be a trap representation just because it gets passed to free().
  Wiedijk: The paper has two different uses of "valid": 6.2.4 and 3.19.3. I would be happy if 3.19.3 said: "Unspecified value, invalid pointer value, or trap representation".
  Svoboda: Is this paper expanding what types can have trap representations, and what did the committee envision things as traps (before this paper)?
  Uecker: Traditionally a trap had undefined behavior from reading it. If you have a non-signalling NaN, you can read it or pass it to a function, so it could not be a trap. Pointer objects can become traps, but a pointer is a value, and that cannot be a trap representation. A pointer expression also cannot be a trap representation.
  Bachmann: The standard does not say much about representations. can you not get rid of "trap representations", which would make many things clearer and simpler?
  Uecker: To me a trap representation is a representation that does not represent a valid value. It is a fundamental concept; we cannot get rid of it. We need a name for them.
  Ballman: I am still confused by "pointer object" vs. "pointer value" vs. "traps" vs. "validity". Pointer objects can be **indeterminate**,

pointer values can be **invalid**. Adding one to a valid pointer is still valid, right? If the pointer is not valid, doing anything with it is undefined behavior.
Uecker: If you can read a pointer object, it is not indeterminate. The paper does intend to change validity. If we want more clarity, we need to adopt the TS.
Straw Poll: Does the committee want something along the lines of N2668 in C2x? 17-1-3 clear sentiment to proceed with this topic
Bhakta: I voted no for both issues I brought up: trap representation and wobbly specification. I am concerned that this paper disallows "wobbly" values without further discussion.

- 7.15 Ballman, The noreturn attribute [N 2700]

  Gilding: MISRA likes this, because it simplifies C++ importing C headers.
  Uecker: My one concern is that this could be combined with "void".
  Ballman: In the future, we could add another variant of noreturn that is a type specifier as the return type for non-returning functions.
  Myers: If the standard library adds an interface, and a program uses their own, using _Noreturn, or they forget to write noreturn entirely on it, that is undefined behavior.
  Ballman: I am fine with that being undefined behavior. If someone includes <stdnoreturn.h> and then uses this, that can be a problem.
  Straw Poll: Does WG14 wish something along the lines of N2700 in C2x? 14-1-5 sentiment to proceed along these lines

Thursday

- 5.12 Ojeda, Safety attributes for C [N 2659]

  Keaton: No questions, we are out of time.
  Straw Poll: Does WG14 wish to adopt something along the lines of N2659 for C2x? 6-10-6 no sentiment to proceed with this

- 5.13 Uecker, improved bounds checking for array types [N 2660] (1 hour)

  Gilding: With regard to sections 2.1 and 2.2, these are already checked by some tools. It is a required check in MISRA.
  Krause: Section 2.2 seems OK, but 2.1 has problems: Standard library function headers have their own declarations…do we change these declarations?
  Myers: With regard to constant declarations, it is hard to provide wording for consistent declarations. Different declarations might be in different scopes. You might have underscores in declaration but not in the implementation for namespace reasons.
  Yodaiken: I am puzzled by what you mean by "consistent declarations"? Should the compiler complain? If a prototype provides bounds, then is it illegal to use without bounds?
  Pygott: Ah, in section 2.1, these are three declarations for the same function.
  Ballman: With regard to section 2.1: Do you have any notion of how much code this could break if it becomes a constraint violation?
  Uecker: It would modify only broken code. Using a newer version of GCC, it catches some problems in my code.
  Ballman: With regard to section 2.2: How does this differ from inserting "static" inside array brackets?
  Uecker: "static" means minimum length, but not maximum length.
  Svoboda: With regard to section 2.1: Array bounds are currently not part of a function signature…are you proposing to change that? What about a function's type? (e.g. I want to declare a pointer to foo, what type should I use?)
  Uecker: It is a good question…probably not. I will need to think about it.
  Svoboda: I do like section 2.2, but it should be an optional compiler warning with no runtime change. CERT rule API05-C advocates VLA's for function signatures, like section 2.2 for static-analysis tool purposes.
  Gilding: With regard to section 2.4, allowing forward references to later arguments is simplest, but leading syntax is confusing and weird.
  Myers: If you allow things out-of-order, you allow a typedef to be shadowed by a later variable.
  Seacord: With regard to section 2.6, would you disallow passing a pointer in this code example here?
  Uecker: No
  Yodaiken: Are you proposing changing standard library functions? Does that make current code wrong, or fail?
  Uecker: No, I would just modify their declarations to include array lengths?
  Svoboda: With regard to section 2.5, strncpy() should have src be greater than or equal to 1. I think the current "static" behavior of allowing an array to be bigger than specified is too useful to ignore.
  Uecker: Right, we keep the behavior of "static" even if we remove that keyword.
  Svoboda: With regard to section 2.6, this is the first code example we teach students because the current behavior is counterintuitive. Besides our course example, what breaks if you change this behavior?
  Uecker: We cannot change the behavior, we can only deprecate usage in counter-intuitive situations.
  Ballman: We had discussions way back that changing pointers to arrays would be an ABI break. I am wondering if that could be a problem here? I think the paper was from Martin Sebor or Gustedt.

Seacord: With regard to section 2.5, we are only providing info to the compiler; could we achieve that with an attribute?
Uecker: Yes that is an option.
Krause: With regard to section 2.5, we can pass bigger destination arrays to strncpy, it just promises never to write past n.
Gilding: With regard to section 3.2, VLA's do not have to live on the stack. The standard allows VLA's to live on the heap.
Gilding: With regard to section 3.3, is there any implementation experience for the ".n" notation in your example?. Adding that would be a major syntax addition.
Uecker: The ".n" notation was based on initializers.
Ballman: With regard to section 3.3, the ".n" is like adding a "this" pointer to C, right?
Uecker: I am not sure I follow you. It accesses the same object on the stack.
Myers: With regard to section 3.4, VLA's already store sizes because the platform needs the size for arithmetic/subtraction. So you do not need new syntax.
Uecker: That is too far. There is no place where the length goes.
Gilding: With regard to section 4.3, I like the first sizeof.
Svoboda: With regard to section 3.2, is type-casting to a VLA (in the example) conditionally normative right now? I would support typecasting VLA's being mandatory, but declaration VLA's on the stack still conditional, or even obsolete.
Uecker: It is optional for now…make the typecasting mandatory, but not VLA's on the stack.
Svoboda: With regard to section 3.3, is this intended to be a flexible array? Are the array contents stored in or after the struct or elsewhere?
Uecker: The array is elsewhere, this is a pointer to an array.

- 5.14 Uecker, maybe_unused attribute for labels [N 2662]

Myers: Does C++ support maybe_unused on labels?
Uecker: Not yet. I will also present this to WG21.
Ballman: This was discussed at the C++ Compatibility SG in May. It was recommended with unanimous content for both WG21 and WG14. I personally support this.
Voutilainen: In what scenario is this useful?
Uecker: Mainly for autogenerated macro-expanded code.
Voutilainen: It is useful for certain templates, then.
Gilding: It is useful for communicating to a compiler or checker to silence warnings. Maybe this should be reworded to say maybe_unused can go on anything and let the compiler decide what it means?
Uecker: I have not thought if that could cause problems.
Gilding: It would be used on a statement or expression to indicate it might be unreachable.
Uecker: I am not planning on writing a paper proposing that.
Bhakta: Is there any implementation experience?
Uecker: Yes, this is supported by Clang and GCC.
Ballman: Do we need to bump the value of has_c_attribute here? Offhand, we probably do not need it; but does anyone else have concerns between minor revisions of C2x? It is about where you can write the attribute without getting a diagnostic. I personally think we are fine not bumping the value.
Uecker: I do not know.
Bhakta: Ballman, I would guess that it should be added just for completion. Normally we do want existing practice in the document, because it makes it easier to vote yes. This may be built for GCC and Clang, but is it used?
Uecker: I used the nonstandard attribute once. I have not researched how much it could be used. It is very new. I will add existing practice into the paper next time.
Meneides: For my nodiscard proposal, we did bump the has_c_attribute value, because we change it to allow a string literal. I do not know if this change is big enough to constitute a bump.
Gilding: The kind of customer that uses QAC is already using maybe_unused, so there is prior use here.
Ballman: Clang's implementation of maybe_unused suggest that we do want to bump the value. I am fine with that being done editorially.
Straw Poll: Does WG14 wish to adopt N2662, with a bump of has_c_attribute, for C23? 20-0-3 so this goes into C23.

- 5.15 Uecker, life time, blocks, and labels [N 2663]

Svoboda: You are documenting the current surprising behavior, not changing or reverting it, right?
Uecker: No, I want to revert the change. The added text constrains the surprising behavior to labels not followed by statements within the block.
Myers: Was in intended before C23 that this example was well-defined? (other questions)
Uecker: I can write another paper to address these other questions.

- 5.16 Seacord, Zero-size reallocations no longer obsolescent feature [N 2665]

Bhakta: Does anyone disagree that undefined behavior is not an obsolescent feature?
Seacord: I do not think any examples of undefined behavior are listed as obsolescent in the standard.

Keaton: I agree. We do not need a rule saying so.
Straw Poll: Does WG14 wish to adopt N2665 into C23? 20-0-0 This goes in.

- 5.12 Ojeda, Safety attributes for C [N 2659]

  Gilding: Great feature, and there is a use case here. It cannot go into the standard because the meaning of "safe" varies.
  Ojeda: The safe attribute is for the function writer.
  Krause: I do not like the term "safe"…it has many different meanings. Example 1 of recommended practice is not safe but marked "safe".
  Ojeda: "safe" is about whether undefined behavior can happen. The example perhaps should be amended to initialize the pointer.
  Voutilainen: This does not match that well with C# "unsafe". In C context (and C++), this looks like a tiny part of a function has no pre-conditions; but does not let me add pre-conditions. It is a tiny part of a system for expressing function pre-conditions, and is not worthwhile to go with just this tiny subset. It would be more worthwhile to add a pre-condition system.
  Ojeda: I see what you mean about pre-conditions, they are about local-correctness more than just undefined behavior.
  Svoboda: Is a signalling-NaN undefined? (Undefined behavior is itself not always well-defined in the C standard). Few functions are not safe. All code I write is [safe] (not to mention bug-free). Seriously, what is the distinction between code that needs vs. does not need this annotation?
  Ojeda: I do write functions with pre-conditions. "unsafe" does not mean you always hit undefined behavior, it means the caller has to ensure right values. All C code is intended to be "safe", but not all individual functions.
  Svoboda: Thread safety is a useful concept; is it part of "safe" or not?
  Ojeda: Yes, "safe" involves thread-safe. Yes, thread-safe code is often slower than thread-unsafe code.
  Svoboda: A "safe-in-single-threaded-code" attribute would be a distinct but useful attribute. Ditto for "async-signal-safe" (e.g. can be used as a signal handler). Making your vector code single-thread-safe is easy. Making it multi-thread-safe is harder. Making it async-signal-safe is probably impossible.
  Ojeda: Agreed.
  Seacord: Making some sort of statement about a property of the function by a user who has not studied the function is a good idea, especially if the compiler can check the property. So this "safe" attribute may be the wrong thing to try to do. I think "safe" is MISRA safety. We need narrower, more verifiable properties. I want to see papers along those lines. One or two of these would be inadequate, perhaps wrapped up in a contract. That would be a good direction for future papers.
  Uecker: The name promises too much. But is also not strong enough. Your definition of "safe" is short and precise but not entirely clear on exactly what it means, such as in vs. out of the context of a single program.
  Ojeda: A function that depends on global state or threads is hard to say. More in terms of modules/APIs. As in Example 3, you need to encapsulate some functions and data. We can bike-shed the name. I adopted the name "safe" from Rust.
  Ballman: In the definition of a "safe" function, we have undefined behavior because we cannot figure out how to define it, but we also have undefined behavior that is left for implementation extension space, and the standard does not distinguish between these two categories of undefined behavior. How would "safe" interact with implementation extensions, without forcing us to enumerate all undefined behaviors?
  Ojeda: It is a good question. Simple answer: it covers both. If you must rely on an extension, you should not use "safe".
  Ballman: You said you want it for the Linux kernel…does that not rely on many extensions?
  Ojeda: True. For those projects, we must see exactly what we can do.
  Yodaiken: Undefined behavior is used both for errors and for extensions. The Rust code will panic when it cannot safely resolve pointer issues, and this makes the Linux kernel unhappy. I think "safe" is not going to get very far. I cited John Regehr's paper about an undefined behavior-free dialect of C. You are at the mercy of non-local effects in your program.
  Ojeda: In Rust we do not have the problem with the kernel, we can take that offline. It is not about making all code safe. If you have C code that calls Rust functions than you can prove the C function is safe.
  Straw Poll: Does WG14 wish to adopt something along the lines of N2659 for C2x? 4-11-6 no consensus to continue

Friday

- 5.4 Gustedt, Lvalue closures v3 [N 2737]

  Gilding: "Lvalue-capture" means "capturing by pointer". A block storage class specifier is far more complicated than lvalue-capture because of automated reference counting. That is not something we would want in C. It is OK to capture a register, but C++ removed registers before adding lvalue-capture.
  Gustedt: The text addresses this, but we do have to treat registers as special.
  Ballman: With regard to registers, this seems odd to me because something being captured has an address, but registers do not.
  Gustedt: Lvalue captures are not capturing the address. The "register" keyword only constrains the programmer, as they cannot take its address. An implementor is OK realizing the register value on the stack.
  Ballman: Personally, I would disallow capturing a register variable until we have implementation experience. We can allow it later.
  Gustedt: That is possible.
  Myers: I would like implementation experience on issues with the proposed wording. There are plenty of issues with wording on "auto" and other stuff this builds on.

Gustedt: I disagree. With regard to auto stuff, we are going as far as we can for C++ compatibility. This is the same for lambdas. We only have to make choices for things that C++ does not address, like registers. There is much work to do.
Myers: This might be worthwhile for C26 or C29, but it is not ready for C23.
Gustedt: We have had nested functions and compound expressions for years and years.
Myers: Nested functions are a complete mess.
Gustedt: Agreed. But there is implementation experience in these fields, even inside C implementations.
Krause: Even if lambdas make it into C23, lvalue-captures go further; they are mostly syntactic sugar…it should not go into the same standard revision as lambda, so we get implementation experience.
Gustedt: I think that is the wrong approach. Nested functions and compound expressions are widely used and require lvalue-captures. Without lvalue-captures, they do not have an easy path. With lvalue-captures right now, you can easily rewrite the header of a nested function.
Yodaiken: I have been asking people what they like to see in C. I hear that they do not want C to follow C++. I also hear that lambdas are great in languages organized around them, but C already has function pointers.
Gustedt: I agree we do not want the complexity of C++ in C. We can profit from the work that C++ has done. We can leverage their implementation experience.
Bhakta: Nested functions are not standardized, not relevant to ISO C. We did not accept a recent nested-function proposal. Implementation is not always easy; it depends on your architecture.
Gustedt: Lambdas have had a wide range of acceptance; nested functions did not.
Gilding: Who is using lambdas / lvalue-capture? Right now it is in public GNU C. We find that customers consequently expect that statement expressions are also part of ISO C, which is incorrect.
Gustedt: Many atomics-library implementations use compound expressions to implement atomic operations.
Wiedijk: This is a weak version of lambdas…too weak to satisfy me. Is this being proposed as required or optional?
Gustedt: Required. There is enough deep language changes that I do not think it can be optional.
Keaton: Do not let chat scroll too much, that is rude to Gustedt.
Bhakta: Users who use GCC and who are used to nested functions and statement expressions are surprised that they are not in ISO C. This is a big difference between portable code and GNU C code.
Gilding: This is a feature with no runtime consequences (as it is currently proposed). In contrast, VLA's affect runtime.
Banham: Defer is wacky to me.
Gustedt: Yes, that is anecdotal. It is a different paper. (N2589).
Wiedijk: Can you imagine a compiler that expands lambdas into lambda-free program?
Gustedt: I have written such a compiler.
Wiedijk: Is that compiler's source publicly available?
Gustedt: I can send you a link. The compiler is not complete.
Straw Poll: Does WG14 want lvalue-capture along the lines of N2737 for C23? 11-7-4 not a strong majority, but general sentiment to go in this direction
Gustedt: I propose that I keep this paper separated from the lambda paper. I would need more feedback on wording…and perhaps a subcommittee for wordsmithing the document.
Voutilainen: Did you consider adding examples for capturing something like a mutex in a lambda? Would that make it easier to understand why lvalue-captures are necessary?
Gustedt: We are not yet far enough for that.
Gilding: You mentioned that you have a public reference implementation. That should be integrated in the next examples.
Gustedt: And the supported syntax is not exactly the same. Send me mail if you are willing to be on a committee to improve this proposal.

- 5.18 Gustedt, Revise spelling of keywords v5 [N 2654]

Ballman: "static_assert" as a keyword already exists in MSVC
Svoboda: Is that because MSVC supports C++?
Ballman: Probably yes. Still it does count as prior art.
Myers: Since these have both been predefined macros, Section 6.10.8 says "any predefined macros start with _ and either an uppercase letter or another underscore". That text needs to be amended to include that these keywords might be macro names. We would also need a statement about not undefining or redefining the various macro names in section 6.1.08.
Gustedt: You are right, we should look into that.
Svoboda: Gustedt, thanks for adding straw poll questions for the document…that saves me from transcribing them.
Svoboda: What about "noreturn"? Or was it delegated to Ballman's paper N2700?
Gustedt: We had no consensus on "noreturn". I only covered items for which we had consensus.
Gilding: There are already compilers using these as true keywords. "Generic" is not on the list as well; and it still has the uglified spelling. because there is no macro.
Gustedt: I took the list of items we had agreed upon. "Generic" as a word itself is probably not the right term. In the C/C++ core I had used "generic-type". We could go in that direction if people want.
Myers: Did we not have a "along the lines of" straw poll in the previous meeting?
Gustedt: In the previous meeting, these were mandatory macros and we produced the list of keywords.

Straw Poll: Does WG14 want to integrate something along the lines of changes 1 – 3 and 5 – 10 as proposed in N2654 into C23? 20-0-2 clear sentiment to proceed along these lines.
Straw Poll: Does WG14 want to integrate something along the lines of change 4 as proposed in N2654 as a new constraint section of Section 6.4.1 for C23? 7-5-10 not clear sentiment
Straw Poll: Does WG14 want to integrate something along the lines of change 4 as proposed in N2654 into the semantic section of Section 6.4.1 for C23? 5-6-10 we do not want to do this
Gustedt: Putting this into the Constraints section requires that a compiler generate a diagnostic on something like "#undef bool".

- 5.19 Gustedt, Make false and true first-class language features v. 5 [N 2718]

  Seacord: In proposed section 6.4.4.5.1p1, "the effect is as if" is a weird phrase to me. Will bool promote to type int (not unsigned int)?
  Gustedt: This is wordsmithing that we can do later. All these types are intmax_t or uintmax_t. Bool will promote to signed int.
  Ballman: With regard to change 8, the "is a zero" part, do the floating-point people have any opinion of if this is clear to them?
  Bhakta: For the complex case this does not seem to work.
  Gustedt: Why? It is a mathematical value zero. We can tweak the wording.
  Gilding: With regard to change 9, the wording is different from C++.
  Uecker: With regard to change 8, should it be a null pointer?
  Gustedt: There could exist architectures where bool has the same width as int. I would entertain accommodating that. C++ always promotes to int, for us it may promote to signed or unsigned int.
  Tydeman: Complex 0 is defined as both parts being 0 so I am fine with that change.
  Straw Poll: Does WG14 want to integrate something along the lines of changes 1 – 4, 6, 7, and 9 – 13 as proposed by N2718 into C23? 20-0-2 clear sentiment to do this
  Straw Poll: Does WG14 want to use something along the lines of the alternative change 8 instead of change 7 as proposed in N2718 for C23? 11-1-10 a lot of abstentions but generally in favor

- 5.20 Gustedt, Introduce the nullptr constant v3 [N 2692]

  Gilding: Allowing nullptr_t is consistent with how to specify types and overload things in general. So this is a useful change.
  Tydeman: With regard to the definition of nullptr: nullptr is a constant but it is not a pointer type.
  Gustedt: But NULL / 0 is also not a pointer.
  Bhakta: Can yo do typeof(nullptr)?
  Gustedt: Yes.
  Bhakta: Then you can get a constant with the same type.
  Gustedt: But it would be an object and not a constant.
  Bhakta: But you can get multiple things that are distinct null pointer. So typeof(nullptr, a) would not work?
  Gustedt: Correct, it would not work.
  Bhakta: This introduces a new class of types (nullptr_t), which is similar but different to void. It is a wider change to C.
  Gustedt: Yes, it makes me a little bit uneasy too.
  Ballman: The printf example differs from C++. In C++ in a vararg context if a type has std::nullptr_t, it gets converted to a void*. We should keep that functionality.
  Gustedt: This is a red herring. For printf() it would work, but if you gave printf an int* where it expects a void*, the type size may be incorrect. I would be uncomfortable with such a change.
  Ballman: I am uncomfortable with the existing behavior because it is a trap. I have a bit less concern if it is a constraint violation, as opposed to undefined behavior.
  Voutilainen: NULL expands to an implementation-defined constant…why did you want it to expand to nullptr?
  Gustedt: I copied that from C++. I do not have a strong opinion on that.
  Voutilainen: I do not think we have enough experience to mandate that.
  Myers: I am concerned about the wording related to constant expressions. I disagree with obsolescing NULL. Writing C code across multiple versions without using obsolescent features is valuable. I am dubious about function parameters of type nullptr_t. It introduces ABI questions. It would be simpler to disallow nullptr_t in function arguments.
  Gustedt: I am confused on obsolescing NULL; if we want to remove it we have to obsolesce it.
  Myers: I do not think we should remove NULL.
  Gustedt: With regard to function parameters, I understand. This makes it more compatible with C++. For some functions you will want to pass a pointer, and a nullptr_t will be a valid pointer. You would always have to cast it to your appropriate pointer type.
  Gilding: I almost want the opposite; I want nullptr_t to be a valid function argument type. Should it also be a valid return type?
  Gustedt: That is possible.
  Straw Poll: Does WG14 want to integrate nullptr along the lines of N2692 into C23? 11-8-1 in favor, but very close
  Gustedt: For naysayers, is this for the rewriting problem?
  Myers: We may want a simpler nullptr feature, but this proposal has a lot of complicated stuff.
  Bhakta: I voted no because I do not see implementations of this in C.
  Gustedt: GCC has __NULL, which is similar; they did not want to use the userspace.

Bhakta: I did not notice multiple C implementations doing this. I also did not see user demand for it.

6. Clarification Requests

The previous queue of clarification requests has been processed.

7. Other Business

The following papers will be deferred to future meetings unless there is time available at this meeting.

7.4 Ballman, __has_include for C [N 2673]

7.5 Gustedt, type inference for variable definitions and function returns v4 [N 2735]

7.6 Gustedt, Simple lambdas v4 [N 2736]

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

7.9 Meneide, Not-So-Magic: typeof(), revision 3 [N 2724]

7.10 Múgica, #warning directive [N 2686]

7.11 Múgica, Identifier - primary expression [N 2687]

7.12 Múgica, Sterile characters [N 2688]

7.13 Gustedt, Improve type generic programming v3 [N 2734]

7.14 Goldblatt, Sized Memory Deallocation [N 2699]

7.16 Krause, @ and $ in source and execution character set [N 2701]

7.17 Tydeman, Overlooked SNAN wording changes [N 2710]

7.18 Tydeman, fmin, fmax [N 2711]

7.19 Tydeman, intbool_t [N 2712]

7.20 Tydeman, Integer Constant Expression [N 2713]

7.21 Tydeman, hypot() [N 2714]

7.22 Tydeman, cr_ prefix [N 2715]

7.23 Tydeman, Numerically equal [N 2716]

7.24 Krause, No function declarators without prototypes [N 2719]

7.25 Krause, Sane C library, when wanted [N 2720]

7.26 Meneide, Preprocessor embed, revision 4 [N 2725]

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

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

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

7.30 Meneide, Transparent Function Aliases, revision 0 [N 2729]

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

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

7.33 Seacord, Volatile C++ Compatibility [N 2743]

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

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

7.36 Thomas, C23 proposal - Annex F overflow and underflow [N 2747]

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

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

7.39 Honermann, char8_t: A type for UTF-8 characters and strings (Revision 1) [N 2653]

8. Resolutions and Decisions reached

8.1 Review of Decisions Reached

Do we wish to replace the October meeting with two meetings starting 30 August and 15 November? 15-0-2
Should we add a 6th day to each of the meetings starting 30 August and 15 November? 10-4-4
Should the committee add Friday, August 27 to the next meeting, making it a 6-day meeting? 9-1-8 I am thinking we should do this.
Should the committee choose Saturday, September 4 vs. Friday, August 27 as part of the next meeting? 1-7-12 Saturday is not desired.
Should the committee choose Monday, September 6 vs. Friday, August 27 as part of the next meeting? 3-7-8 So previous Friday wins.
Does the committee wish to adopt the Core proposal from N2683 into C23? 13-0-5 Core proposal goes into C23.
Does the committee want the "ckd_" identifiers from N2683's Core proposal in future library directions to be potentially reserved identifiers? 18-0-0
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
Does the committee support, in principle, the approach of defining a small number of alternative options for uninitialized-read semantics, allowing users to choose as they wish? 6-9-6 no sentiment to pursue
Does the committee support, in principle, the approach of explicitly permitting, at the implementation's per-instance choice, either a compile-time error, a non-silent runtime error, or some other more-defined-than-undefined behavior for uninitialized-read semantics. 10-9-3 not really sufficient
Does the committee want to make uninitialized reads of scalar non-character typed objects of automatic storage duration always be undefined behavior? 10-9-3 not enough direction to say people are definitely in favor
Does the committee want type-generic lambdas along the lines of N2738? 10-6-5 general sentiment in favor, maybe we want lambdas but not along the lines of N2738.
Does WG14 wish to adopt N2709, proposed wording alternative one, into C2x? 14-2-5 This goes into C23.
Does WG14 wish to adopt N2650 into C23? 18-0-2 This goes into C23.
Does WG14 wish to adopt the change to 7.12.7.3 listed in N2651 into C23? 20-0-3 so it goes in.
Does WG14 wish to adopt the change to 7.12.11.1 listed in N2651 into C23? 17-0-6 so it goes into C23.
Does WG14 wish for CFP to create a new revision of TS 18661-5 based on C23? 19-0-2 there will be a revision of Part 5.
Does WG14 wish to accept N2670 into C23? 19-1-3 so it goes into C23.
Does WG14 wish to accept N2671 into C23? 16-0-2 So it goes into C23.
Does WG14 wish to accept N2672 into C23? 21-0-1 So it goes into C23.
Does the committee want something along the lines of N2668 in C2x? 17-1-3 clear sentiment to proceed with this topic
Does WG14 wish something along the lines of N2700 in C2x? 14-1-5 sentiment to proceed along these lines
Does WG14 wish to adopt something along the lines of N2659 for C2x? 6-10-6 no sentiment to proceed with this
Does WG14 wish to adopt N2662, with a bump of has_c_attribute, for C23? 20-0-3 so this goes into C23.
Does WG14 wish to adopt N2665 into C23? 20-0-0 This goes in.
Does WG14 wish to adopt something along the lines of N2659 for C2x? 4-11-6 no consensus to continue
Does WG14 want lvalue-capture along the lines of N2737 for C23? 11-7-4 not a strong majority, but general sentiment to go in this direction
Does WG14 want to integrate something along the lines of changes 1 – 3 and 5 – 10 as proposed in N2654 into C23? 20-0-2 clear sentiment to proceed along these lines.
Does WG14 want to integrate something along the lines of change 4 as proposed in N2654 as a new constraint section of Section 6.4.1 for C23? 7-5-10 not clear sentiment

Does WG14 want to integrate something along the lines of change 4 as proposed in N2654 into the semantic section of Section 6.4.1 for C23? 5-6-10 we do not want to do this
Does WG14 want to integrate something along the lines of changes 1 – 4, 6, 7, and 9 – 13 as proposed by N2718 into C23? 20-0-2 clear sentiment to do this
Does WG14 want to use something along the lines of the alternative change 8 instead of change 7 as proposed in N2718 for C23? 11-1-10 a lot of abstentions but generally in favor
Does WG14 want to integrate nullptr along the lines of N2692 into C23? 11-8-1 in favor, but very close

8.2 Review of Action Items

Keaton: will investigate problems with the WG14 reflector. (done during meeting)
Keaton: Put standardization schedule as an agenda item on the full January agenda.
Tydeman: Repsond to Austin Group about whether the remquo result should be a NaN.
Bhakta & *Ballman*: Have CFP SG target a paper to WG21 summarizing changes to C floating-point.
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.
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.
Keaton: See what actions we need to take with ISO in order to start the TS 18661-5 revision. (This is done.)
Keaton: Get SC22's permission to revise TS18661-5.

9. PL22.11 Business

10. Thanks to Host

10.1 Thanks to ISO for supplying Zoom capabilities

11. Adjournment (PL22.11 motion)

Moved by Tydeman, seconded by Seacord, objections? (none)
Meeting adjourned on Friday, June 18, 2021 at 16:50 UTC.