Draft Minutes for 8 – 12 March, 2021

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

WG 14 / N 2690

Dates and Times

Each day will have a half-hour break from 16:00-16:30 UTC.

8 March, 2021	14:30 – 18:00 UTC
9 March, 2021	14:30 – 18:00 UTC
10 March, 2021	14:30 – 18:00 UTC
11 March, 2021	14:30 – 18:00 UTC
12 March, 2021	14:30 – 18:00 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. Gilding will take minutes on Thursday if Svoboda is unavailable, and when Svoboda led the discussion.

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

Name	Organization	NB	Notes
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	
Barry Hedquist	Perennial	USA	PL22.11 IR
Tommy Hoffner	Intel	USA	
Rex Jaeschke		USA	Invited Guest
David Keaton	Keaton Consulting	USA	Convener
Will Klieber	CERT/SEI/CMU	USA	
Philipp Krause	Albert-Ludwigs-Universität Freiburg	Germany	
Kayvan Memarian	University of Cambridge	UK	
JeanHeyd Meneide	NEN	Netherlands	
Maged Michael	Facebook	USA	
Joseph Myers	CodeSourcery / Siemens	UK	
Miguel Ojeda	UNE	Spain	Spain NB
Clive Pygott	LDRA Inc.	USA	WG23 Liaison
Robert Seacord	NCC Group	USA	
Peter Sewell	University of Cambridge	UK	Memory Model SG
Peter Sommerlad		Switzerland	Invited Guest
Nick Stoughton	The Austin Group	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		Finland	Invited Guest
Freek Wiedijk	Plum Hall	USA	
Michael Wong	Codeplay	Canada, UK	WG21 Liaison
Jörg Wunsch		Germany	Invited Guest

1.3 Procedures for this Meeting (Keaton)

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 2628] (WG 14 motion)

Moved by Stoughton, Seconded by Pygott
Tydeman: Sent errata to the scribe.
Svoboda: Got them, thanks, Fred!
No objections

1.6 Review of Action Items and Resolutions

Stoughton should be listed as representing The Austin Group

1.7 Approval of Agenda [N 2666] (PL22.11 motion, WG 14 motion)

Keaton: Is there a PL22.1 motion to approve the agenta in N 2678?
Moved by Seacord, seconded by Ballman. Objections? (None)

1.8 Identify National Bodies Sending Experts

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

1.9 INCITS Antitrust Guidelines and Patent Policy

1.10 INCITS official designated member/alternate information

Keaton: Please see PL22.11 chair (Bhakta) if you see an error in this information.

2. Reports on Liaison Activities

2.1 ISO, IEC, JTC 1, SC 22

Keaton: The JTC1 chair is our friend, and advocates issues for us. They helped eliminate some limitations on technical corrigenda.

2.2 PL22.11/WG 14

Hedquist: Are we taking cash23?
Keaton: We will still have the provenance TS in-flight when C23 is published. We will need to revise the standard after C23 at some point, as we always do. This means a new revision in about 5 years, instead of 10 years as we have done historically.
Hedquist: We need to define "soon". That is, when will the next revision of C come out?
Myers: Also will the next revision focus on bugfixes or new features?
Keaton: We should think about this and bring it up in this section of the June meeting.

2.3 PL22.16/WG 21

2.4 PL22

2.5 WG 23

Pygott: We are working with ISO to publish our documents free of charge. ISO says they are Technical Reports (TR's) not International Standards (IS's), so we are trying to convert our documents to International Standards.
Keaton: Editions 1 & 2 of Vulnerabilities TR were free, but then ISO changed the rules, so WG23 is working around that change.

2.6 MISRA C

Gilding: We have not decided how it will be numbered.

2.7 Other Liaison Activities

3. Reports from Study Groups

3.1 C Floating Point activity report

Bhakta: There is a proposal for an update for part 5, so we will be proposing that next meeting.

3.2 C Memory Object Model Study Group

Sewell: We are thinking about the next step beyond the current, working draft TS. We will discuss on Wednesday.

3.3 C and C++ Compatibility Study Group

Organizational Information [N 2627]
Omnibus of WG21 Papers (Feb 2021) [N 2656]

Ballman: First meeting was in February, 2nd meeting was last Friday. There is a 9 months' backlog of papers, we will most likely review 2 per meeting. We are currently focused on things already in C23 & C++23. Our next priority are things expected to get in to either release. We have 2-4 people from WG14, but we could use more. Our next meeting is in April.

4. Future Meetings

4.1 Future Meeting Schedule

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

14-18 June, 2021	Virtual, 13:30-17:00 UTC each day
4-8 October, 2021	Minneapolis, Minnesota, US (tentative)
31 January- 4 February, 2022	Portland, Oregon, US (tentative)
11-15 July, 2022	Strasbourg, France (tentative)

(Note: October, 2021, will become virtual if the global health situation is still uncertain as of the June 2021 meeting.)

Pygott: Are the times in the June meeting correct?
Keaton: They are different from this meeting, but those on daylight savings time will not notice any difference.

4.2 Future Mailing Deadlines

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

Post-Virtual-202103	2 April 2021
Pre-Virtual-202106	14 May 2021
Post-Virtual-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

Gustedt: Can we have a strategy about how to proceed papers given more slack to editors?
Keaton: We progress papers by going through the latest version of each paper, either voting it in or declaring it to not be ready for inclusion in the standard. January is the deadline for getting existing papers into C23, October is the deadline for new papers. I would be against giving more slack to editors; when we vote something in, we want the document voted in as is with minimal editing.

5. Document Review

Monday

- 5.1 Working draft updates
  - Meneide, C2x Working Draft [N 2596]

    Meneide: This mainly has typo fixes. There should be a diff-mark from last draft, but it is currently broken. I still need to add N 2597.

  - Meneide, C2x Working Draft - Editor's Report [N 2598]

    Meneide: I am still building a new paper submission system for us, which should alleviate the ability to submit / index / search papers.
    Myers: If some editorial change was made that I disagree with, can I file a Gitlab issue? What should I do?
    Meneide: That is covered in N 2598. I take a hyper-conservative approach with it. You can file a Gitlab issue or email me to bring my attention to issues.

- 5.2 Meneide, Not-So-Magic: typeof() for C [N 2619]

  Gilding: I like the "remove_quals" operator. It is a great usability improvement.
  Krause: The naming seems inconsistent. So remove_quals should only work on type names?
  Meneide: This is doable. But it would decay arrays that people wanted to keep occasionally.
  Krause: Why do we need typeof on types?

Meneide: Mostly for passthrough.
Gustedt: I do not think that we need the remove-qualifier version
Meneide: Some think we can do an lvalue-conversion to strip qualifiers. If you lose lvalue-conversion you lose array qualifiers. People had to do complex macro machinery to strip qualifiers off of types, which we resolved by adding remove_quals.
Ballman: With "auto", I have to assign the expression to a new "auto" variable. The remove-quals operator saves me from copying objects, which makes it worthwhile for me.
Bhakta: In 4.2.6 example 2, there is an error.
Meneide: Ballman pointed out this error to me, so it is fixed in the latest draft.
Krause: Not allowing remove_quals on expressions is not the same as removing types.
Gilding: Existing operations like sizeof (and current typeof implementations) seem to take a flexible approach: If it can take a type it can take an expression.
Meneide: I did want to emulate sizeof as much as possible.
Bhakta: I recommend that you propose the votes you want and then discuss other votes.
Gustedt: The problem with remove_qualifiers is: no prior art. (On the other hand, there is lots of prior art for typeof).
Ballman: Prior art is that existing implementations of typeof are inconsistent. Some remove qualifiers, while others do not.
Meneide: Other implementations treat qualifiers differently.
Goldblatt: The C++ case is instructive: They added remove_qualifiers as a library feature, and consider it an important addition.
Meneide: C++ has ways of matching on types & qualifiers that we lack.
Svoboda: Perhaps we should move remove_qualifiers into its own proposal or at least its own poll?
Meneide: This opening poll is to get the text in the door.
Gustedt: But there are multiple open options.
Myers: There are enough issues with this paper that we should see a new version before the final vote.
Straw Poll: Does the committee wish to adopt something along the lines of N 2619 into C23? 18-0-0
Straw Poll: Does the committee wish to use a "_Typeof" keyword with the usual header for the typeof feature in N 2619? 7-7-5
Straw Poll: Does the committee wish to use a "typeof" keyword for the typeof feature in N 2619? 16-2-1
Ballman: In the chat window before the break, Stoughton left.
Straw Poll: Does the committee wish to use a completely new keyword (rather than typeof or _Typeof) for the typeof feature in N 2619? 1-14-3 Clear preference
Straw Poll: Does the committee wish typeof to accept type names (in addition to expressions) as a valid argument in N 2619? 17-1-4
Straw Poll: Does the committee wish remove_quals to accept expressions (in addition to type names) as a valid argument in N 2619? 11-2-5

- 5.3 (FKA 7.3) Blower, Add support for preprocessing directives elifdef and elifndef [N 2645]

Gilding: This is trivial to implement.
Bhakta: We try to keep the preprocessor stable. We do not see a problem with the current approach.
Ballman: Yes, this does not introduce a new way to do something previously non-doable. It makes C a more teachable & approachable language.
Bhakta: No C implementations of this feature are listed in prior art. I have not seen a need for this.
Wunsch: We have the defined() operator so I would prefer to deprecate #ifdef rather than introducing these new keywords.
Wiedijk: I like this because it makes the language more orthogonal & nicer.
Bagnara: Jones in 2005 suggested adding this to the language. There was a preprocessor called "mpp" which provided both new directives.
Gilding: Deprecating ifdef as Wunsch suggests would also make the language orthogonal.
Ballman: Deprecating #ifdef is implausible because they are used everywhere.

Bhakta: Was not one of the guidelines to C to have only one way of doing things?
Ballman: Yes that was in the charter. The preprocessor has always violated this guideline.
Wunsch: I am strongly opposed. I prefer to save keystrokes.
Goldblatt: I found one case of a C standard library implementation removing a commit of an #elifdef to their code repository.
Meneide: I got emails from people excited that we might fix this.
Ballman: Blower posted to the chat an email from someone who is excited about this feature.
Bhakta: I disagree with the attitude of adding something to C because it is "small".
Gilding: Non-C preprocessors like yacc have this. I think that should count as prior art.
Ballman: There is a Unix tool that provides a cheap alternative to the C preprocessor that supports #elifdef.
Jaeschke: Is there any feedback from the C++ Liaison group yet?
Ballman: This paper is in their backlog. Assuming WG14 likes this paper I will propose it to WG21.
Gustedt: It would be good to coordinate with WG21 / C++. If we accept this, we should accept it conditionally with WG21.
Krause: I cannot imagine WG21 would have a problem with this.
Meneide: WG21 already has a preprocessor directive we do not have. As long as there is intent to port this kind of feature to both committees, I do not see a reason to delay approving this feature.
Straw Poll: Does the committee wish to adopt N 2645 into C23? 15-1-4 This goes in to C23.
Ballman: I will make sure that the C++ compatibility group examines this paper.

- 5.4 Sommerlad, Make assert() macro user friendly for C and C++ [N 2621]

  Gilding: I am concerned about the feasibility of the 2nd example.
  Uecker: This is a problem with all macros. We recently fixed a similar problem with offsetof(). Adding this just for assert() would make the language more irregular.
  Myers: Seconded. This is a special case for one macros, when many other macros share this property.
  Bhakta: I would like to hear your presentation on the rest of the paper.
  Meneide: I prefer that we fix this now, despite it being a specific instance of a general problem.
  Gilding: We should not reject this just because it does not fix the general problem. We need time for a more general fix and we should try to find one. A "..." is not a suitable extension for a single argument.
  Ballman: My problem here is that re-specifying the argument list to be varargs shows up in surprising places that impact users. That is, this fix exposes semantics we do not intend to expose.
  Bhakta: This cases an assert() (with no arguments) to no longer give you an error diagnostic. So I do not think this change is good.
  Gilding: Just add a constraint to 7.2.1.1 about an assignment expression.
  Straw Poll: Would the committee wish to adopt something along the lines of N 2621 into C23? 7-3-8
  Svoboda: It is a good problem to solve, but much bigger than assert(). More attention is needed.
  Ojeda: I do not know if we will encounter some other similar problem.
  Myers: I would like a language-level solution, not just a solution for assert().
  Wiedijk: I do not see the urgency of the problem; it does not prevent people from writing correct assert() code.
  Ballman: Having implementation experience would go a long way to make me comfortable with this solution.

Tuesday

- 5.5 Seacord, Specific-width length modifier [N 2623]

  Gilding: What is the use case for including fastest specific-bit types?
  Seacord: The use case is for extended integer types, because they cannot accept precise widths for now.
  Myers: Suppose we pass a type int_fast16_t or int_least16_t > 16bits? Consistency says we should only convert the value passed to the corresponding type (int_fast16_t) The wording says "shall be converted to N bits" when it says "shall be converted to the unpromoted type".

Seacord: Agreed, this wording is faulty.
Hoffner: If you do not know the exact storage size you cannot tell it how many bits to read.
Seacord: If the format conversion specifier uniquely identifies the type being passed, then the library implementation will know how many bytes of storage are needed to represent the type. The conversion specifier is meant to match the type, not be a cast.
Ballman: If the fast type is 16 bits and underlying type is 32 bits, what's wrong with the wording?
Seacord: Each of these types (fast/least) has an actual size and on a platform, users expect to use all the values of the underlying type, and that is what should be printed by these functions.
Myers: The reason we have "shall be converted to" wording: If you pass a short to %hd, the value is going to remain in the value.
But what happens if you use %hd but you pass an int (not a promoted short)?
Gustedt: You should really take the wider type. The valid values of the wider type could have undefined behavior when seen by the printf() function.
Krause: I prefer converting back to unpromoted type. Do we really need a new paper? s/shall be converted to N bits/shall be converted to an unpromoted type/;
Seacord: I will make that change and present an update later this week.
Keaton: Agreed.

- 5.6 Ballman, Digit separators (updates N 2606) [N 2626]

  Svoboda: Are there really no constraints on the 's? So both 123'456'789 vs 123'456'78? are permissable?
  Ballman: Yes, they are unconstrained except that you must start with a digit.
  Wunsch: Is it permitted to have the ' after an initial "0x"? Perhaps negative examples would be allowed?
  Krause: I do not want to see too many examples.
  Myers: In the syntax for pp-number, you are saying that a separator cannot be followed by a universal character name. Your syntax is consistent with C++. Why can C++ have non-digits after a separator?
  Ballman: I do not know why.
  Straw Poll: Does the committee wish to adopt N 2626 with editorial changes into C23? 17-1-0 it goes in

- 5.7 Svoboda, Towards Integer Safety (updated from Oct meeting) [N 2629]

  Svoboda: Most recent update:

  - Updated after Myers's comments on normative wording
  - Safe ints now apply to everything other than bool, plain char, and enum
  - Specified that the result must be a modifiable l-value

  Svoboda: Next set, top three changes are the major ones:

  - Usual arithmetic conversions section is biggest clarification that the new types follow the behavior of the C builtin types
  - Agreed to change to a new header rather than put this in stdlib.h
  - Switched the meaning of the flag to match GCC's existing practice remaining changes are not controversial.

  Bhakta: Process question: with regard to the timing of updates, this update was released after the paper; are we allowing late or last-minute updates? Reviewers need time to look at documents.
  Keaton: Two issues: 1: Usually discussing updated versions of papers is OK. 2: The two week rule applies. This paper was published less than two weeks ago so it is now OK to object. ISO allows updates up to two weeks before publication.
  Bhakta: So how late is allowed?
  Keaton: Two weeks before. So we can discuss the update, but will not if you want to object.
  Bhakta: I want to discuss the content but am mindful that we need time to review.

Keaton: There is similar reasoning to the agreed homework item for Seacord.
Bhakta: Let's discuss the content but defer the straw poll? Compromise?
Keaton: This is a great compromise. OK to vote on Friday?
Bhakta: Yes.

Svoboda: (page 7) arithmetic conversions - a question from October was how these are handled in operands. Only example a3 gets it right. This is a problem shared with the builtin operators.
GCC uses infinite precision for the operation, and knows the size of the result, no implicit conversions apply to the operation. We use the same wording as GCC and get the correct behavior.
Using the two-argument form results in a compile-time error because the strongly-typed result cannot assign to the wider result.
We changed how the normative text works "under the hood", importing the wording from GCC because it is already correct.
The example with nested builtin arithmetic operations is still not magically self-fixing (and would not be with GCC)
The largest change was adding the virtual infinite precision type.

Krause: are a and b both integer types or arithmetic? And should this be mandatory for freestanding implementations?
Svoboda: This is answered by paragraph 3 which specifies integer types. What do you mean by freestanding?
Krause: It is defined in the Standard. A freestanding implementation does not need to provide everything a hosted implementation does. Is this in the list of optional features?
Krause: a and b are referred to as objects of integer type in paragraphs 1 and 2, but as types in paragraph 3.
Keaton: They contain values <u>of</u> integer type.
Svoboda: This change needs to be duplicated throughout the document. I prefer wording that just refers to type2 and type3 rather than a and b. I had not considered the freestanding question. This needs a separate CR.
Gustedt: More generally the wording is improper; you cannot use "must". "In other words" is wrong, text should either be normative or non-normative and should be in a footnote if not.
Should there be a constraint on type2 and type3?
Svoboda: I will change to "shall be".
Gustedt: That makes it undefined behavior, we would need a Constraints section. We should type check these if we can do so.
Gilding: Do we expect this feature to be implemented inline with single/few instructions, or do we expect it to need a support library like GMP? Needing a backing library makes it "obviously" unsuitable for freestanding.
Svoboda: This really depends on the platform and the optimizer. I would expect most operations to inline, but some might need more complicated flag checks.
Myers: Constraints are difficult to diagnose. Existing practice is <u>not</u> to use constraints in the library clause. Constraints require a diagnostic.
Svoboda: I would really like type safety here. Is this difficult for any platform to enforce? The proof of concept implementation uses `_Generic`; no compiler magic is needed.
Myers: Actually `_Generic` may match enumerations when they're compatible to an integer type. It is very hard to rule them out perfectly. Some other things are easy to rule out (such as floating types).
Svoboda: Some things can be constrained. Maybe leave some holes for the very imaginative expert user to uncover undefined behavior?
Banham: Are the overflow flags for add and subtract equivalent to carry/borrow? Can we use this to implement wide accumulators?
Svoboda: Yes, they are. You can implement e.g. a 128-bit add with this.
Banham: Should the text clarify this?
Svoboda: In the normative text?
Banham: e.g. the infinite precision earlier is not really infinite - it relies on status flags.
Svoboda: The Standard does not give creative examples, that's for tutorials.

Banham: Generics can set the precision of the result to the precision of the output type.
Svoboda: This has problems if the result type is smaller than the operands. The wording taken from GCC avoids these kinds of problems. Overflow flags are not really "in" C. Are you asking for clarity about the "infinite" type?
Banham: I am concerned that this leaves room for implementor divergence.
Gustedt: With regard to constraints, we do not have them in the library, but we want to give the user more tools here to avoid undefined behavior, and it seems silly to introduce new, completely avoidable undefined behavior here. I would like something stronger to have the types constrained.
Svoboda: Are there any existing examples we can copy in the library?
Gustedt: No.
Bachmann: Overflow and carry are not the same thing for signed integers. One cannot directly use them to implement multi-precision.
Svoboda: I assume I would use two unsigned 64-bit integers to implement 128-bit math; one cannot use the checked 128-bit type if it does not exist on the platform as a builtin. If the platform does have it, then it should be added. Note that GCC's 128-bit integers do not support all builtin operations. "Overflow" has overloaded meaning…overflow for signed, carry for unsigned. We used the "inexact" flag to mean both here. Overflow is impossible with infinite value range, only on the cast back to the result type. Overflow is not used in the normative text for this reason.
Keaton: Trying to have this mean carry is a mistake…it has different meanings for signed and unsigned. It would be over-specification to talk about this in the normative text.
Gilding: tgmath.h also has the constraint problem; it uses undefined behavior. Is this precedent for the idea that adding constraints might improve the library section in general?
Uecker: Type-generic macros look like regular functions. Should not they be marked out as different?
Svoboda: I avoided that wording as I do not want the generic function issue to hold up progress on this library.
Myers: Undefined behavior in tgmath.h is also useful for extensions - an implementation is free to also support complex extensions, such as complex functionality for a function that only specifies real arguments, or to convert the argument if not. There are reasonable interpretations of some operations. This is similar to "checked"…it can often be clear what the logical result of an operation may be and undefined behavior allows scope for it to be provided as an extension, rather than requiring a diagnostic.
Bhakta: I do know of implementations that extend the library like this - this latitude is used. I do not want to backdoor constraints into the library section! We would need a separate paper for that. There are real use cases for a library to have macro implementations that take advantage of the undefined scope for extension. This idea is a radical change that would break implementations.
Svoboda: That would be a topic for a separate paper.
Gustedt: An alternative option would be to add a recommended practice note that the implementation should diagnose unsupported argument types.
Keaton: That has no bad implications for the rest of the library.
Svoboda: Let us table the recommended practice. Need normative tweaks to the integer type wording. I will bring this back on Friday.
Keaton: I am hearing significant objections to the idea of adding constraints to the library; so I recommend adding wording tweaks such as "shall" and bring this paper back later in the week.
Seacord: Procedural issue: Dan changes the document numbers only on weekends. If a new document number is needed mid-week (i.e. mid-meeting), you need to ask Keaton instead.
Svoboda: No vote

- 5.8 Wunsch, C23 proposal: formatted input/output of binary integer numbers (rev. 3) [N 2630]

  Straw Poll: Does the committee wish to adopt N 2630, without recommended practice, into C23? 13-0-1 passes
  Krause: Why do we not have "B" for binary I/O when we have uppercase 'X' for hexadecimal I/O?
  Wunsch: We do not need "B". The case difference is for digits that are letters, as hexadecimal has but binary

does not.
Straw Poll: Does the committee wish to adopt the "recommended practice" in N 2630 into C23? 13-2-3 passes

- 5.9 Ojeda, secure_clear [N 2631] (1 hour)

  Tydeman: Would making the parameter "s" be volatile help or not?
  Ojeda: I do not remember.
  Ballman: Specifying "volatile" makes it unclear whether each byte is specified once only, or in a group. So SG22 elected not to recommend "volatile".
  Bachmann: As for the signature, if it is called memset_<suffix>, it would be nice to have the same arguments as memset(). Those who do not care about the "c" parameter can put in whatever they like. Those who do care can put in 0. So why have the "c" parameter?
  Voutilainen: If the "c" parameter is removed, it becomes questionable whether memset_explicit() is the right name.
  Gustedt: Implementation-defined values would be written. Could these always be the same values for one call; could they repeat? Would stating "random" in the standard be allowed?
  Voutilainen: The implementation would be allowed to write anything.
  Gustedt: That is the meaning of "unspecified".
  Voutilainen: Implementations are still required to document what they do.
  Krause: memset_explicit() uses the same interface as memset(). This solves the problem at hand.
  Ballman: I was in favor of the "c" parameter. But the SG22 meeting makes me think it is a lie. We cannot describe the effects of the abstract machine. I think it makes sense to drop that parameter and allow implementations to write whatever they want.
  Voutilainen: Writing 0's does solve the problem. Users would expect what they pass as "c" gets written. If you allow implementation-defined semantics, then just remove the parameter.
  Svoboda: Platforms should not write whatever they want. They could write *etc/password*.
  /*Voutilainen*: You need to trust your implementation to do reasonable things.
  Svoboda: If I trusted my implementation, I would just stay with memset().
  Voutilainen: memset_explicit() is imperfect, you must have some trust in the platform.
  Svoboda: Agreed. But if I'm using memset_explicit() I have low trust and want more control over what gets written.
  Krause: This is not the original problem I wanted to solve here. I just wanted to overwrite my old secret, but perhaps use the memory again later.
  Bachmann: I agree with Uecker. If we choose to ignore "c", we have to change the signature. I am not in favor of changing the signature, and change the function name.
  Gilding: An implementation might clear data but might not clear the secret, because it might have been copied or exist elsewhere.
  Svoboda: Krause, if you want to keep your cleared memory around, memset() is good enough. Gilding, memset_explicit() does not handle copies, (part of being unenforceable). We are only solving the part of the problem that we can.
  Krause: There could be a delay between when memset_explicit() is called and memory is cleared.
  Svoboda: Yes, the paper does nothing to address timing or race condition issues.
  Straw Poll: Does the committee wish to replace the "c" parameter with a specific value, from memset_explicit() in N 2631? 4-8-11
  Straw Poll: Does the committee wish to replace the "c" parameter with "implementation-defined values" from memset_explicit() in N 2631? 4-9-8
  Ojeda: Does the committee wish to use Alternative 1 with implementation-defined semantics in N 2631?
  Bhakta: You are asking "What this function does is implementation-defined".
  Gilding: This gives implementations the latitude to keep the secret in caches, right?
  Ojeda: Yes
  Voutilainen: That would be the advantage of implementation-defined semantics.

Gustedt: All we can provide is intent. Implementations are allowed to document whether they clear a cache, but we cannot enforce that.
Ojeda: The idea was to require the implementation to document its behavior.
Krause: Yes, "implementation-defined" requires documenting its behavior.
Ojeda: In Alternative 2, "the implementation is encouraged to document its behavior".
Keaton: I suggest we vote on the three options suggested. You can produce revised options on Friday if you wish.

Straw Poll: Does the committee prefer Alternatives 1, 2, or 3 as is in N 2631?
Alt 1: 13
Alt 2: 4
Alt 3: 0
Alternative 1 wins

Voutilainen: Alternative 2 would be problematic to apply to C++.
Bachmann: We still have recommended practice.
Ojeda: Do we still want the recommended practice?
Straw Poll: Does the committee prefer removing the Recommended Practice section from Alternative 1 in N 2631? 15-0-7 passes
Bhakta: What we voted with regard to freestanding before was just the functions, not to any functions added since the vote.
Straw Poll: Does the committee prefer adding memset_explicit() to the exception list in <string.h> freestanding implementations? 3-8-10 So memset_explicit() is not added to the exception list and will be required in freestanding implementations.

Wednesday

- 5.10 A Provenance-aware Memory Object Model for C
    - Sewell, Introduction for discussion of N 2577, Working Draft Technical Specification [N 2624]
    - Sewell, Working Draft Technical Specification [N 2577]

    Gilding: Is the upcoming vote a typical straw poll? (Yes!)
    Straw Poll: Does the committee wish to use N 2577 as the base document for TS 6010? 20-0-2
    Sewell: Are there any reasons for the abstentions?
    Ballman: No particular reason
    Bhakta: I was waiting on feedback from someone else at IBM.
    Sewell: Please pass that feedback on to me as soon as you get it.
    Gilding: We had some dissent at my company; some people view sub-object provenance as useful.
    Sewell: Keaton, are we happy with a discussion without a paper at this point?
    Keaton: It would be helpful to have a document at some point. Please summarize the discussion into a document.
    Svoboda: This next document is not N-numbered, right?
    Sewell: Right. It is available at:
    https://htmlpreview.github.io/?https://github.com/C-memory-object-model-study-group/c-mom-sg/blob/master/notes/built_doc/cmom-0006-2021-03-08-clarifying-uninitialised-reads-v5.html
    Gilding: Trap representations apply to machines that might have "initialized" flags on memory.
    Seacord: Trap representations have not worked out, and we would be better off without them. Most types do not have any trap representations. (e.g.: trapping on an uninitialized read). They raise questions of undefined behavior (based on trap representation vs. independent value). Perhaps we should acknowledge that trap representations are undefined behavior?
    Krause: Hidden bits in the hardware can be used for trap representations.
    Svoboda: Uninitialized reads is a case of bounded undefined behavior, according to Annex L. Also the

standard allows all types to have trap representations **except** unsigned char.
Gustedt: The wording "trap representation" is not good for today. But the concept itself is still important for invalid bit configurations. For example extra bits on a bool will create undefined behavior.
Sewell: Using a bad representation of type bool might result in arbitrary behavior.
Gustedt: GCC does that, sometimes it looks for the parity bit. So this makes your program completely inconsistent.
Sewell: So we need a concept akin to trap representations.
Bhakta: We do use trap representations in other types, but not integers. Do not get rid of trap representations in integers, they are a big part of C. We have a fixed-point type that uses trap representations. I disagree that trap representations are a mistake.
Sewell: Are such trap representations undefined behavior if they are used rather than loaded?
Bhakta: No, we have several unused bits in our fixed-point type. The first bit is the sign, and other values are non-canonical. We let extra bits pass through, so we dont use them as trap representations, but our customers use them.
Uecker: I agree with Bhakta…we need trap representations. Unsigned char is not explicitly stated as non-trap representations. If integer types have trap representations, there are non-visible bits in the abstract machine. Many systems, such as valgrind, will use non-visible bits for housekeeping, these can also be trap representations.
Seacord: The purpose of trap representations was to detect uninitialized reads. If all values cannot have trap representations, that defeats this purpose.
Sewell: I disagree that was the purpose of trap representations. The purpose was to identify bit patterns that do not represent valid values.
Seacord: Clearly bool has trap representations since it takes a byte. Reading a bool never causes a trap, then that value causes undefined behavior.
Gustedt: What trap representation means today is not a good name for the feature. What we need now is "incorrect bit pattern for this type", which is today's trap representation.
Seacord: My big problem is that trap representations lead to the notion that any read can be undefined behavior.
Uecker: We need something that indicates invalid values (for object types). What we have in the standard makes sense. Trying to describe behavior for trap representations is a huge can of worms.
Sewell: We should look at instances of "unspecified value" in the standard as well before moving on.
Keaton: There are two reasons we had trap representations: 1: to support hardware, 2: the language-independent arithmetic standard included traps. There was a big ISO push for standards to reference each other. There is no longer this pressure, so we can revise Annex H to no longer support traps.
Svoboda: s6.2.6.1p5 says: "Certain object representations need not represent a value of the object type. If the stored value of an object has such a representation and is read by an lvalue expression that does not have character type, the behavior is undefined. If such a representation is produced by a side effect that modifies all or any part of the object by an lvalue expression that does not have character type, the behavior is undefined. Such a representation is called a trap representation."
Goldblatt: Undefined behavior and trap representations can be used for debug checking, a la valgrind or Clang's Undefined Behavior Sanitizer (UBSan). It would be good to get feedback from these tools. These tools issue warnings at time of uninitialized value load. If the load becomes allowed, we would need to defer warnings to later, when a load gets used.
Gilding: That sounds like two concepts in trap representation: "trap object state" vs. "non-value representation". Perhaps we should separate them?
Sewell: I agree, although discussing unspecified values should help. People seem happy that we might revisit trap representations and perhaps change terminology. There are some cases for some types where hardware must be allowed to trap on load.
Svoboda: How can the 2nd code example be undefined behavior if it has no trap representations?
Gustedt: The int can be stored in a register that has a trap state.
Svoboda: …which would be outside the int, which has no trap representations.
Keaton: Myers posted a link to DR330 which addresses this. Clarification: This is about the IA64 bits

Sewell: I would guess that Martin Sebor says this is undefined behavior so I can get my compiler to report errors. But that is a heavy-handed mechanism of undefined behavior for error detection. We could say "either the compiler has to report this" or "I must get a reasonable unspecified value".
Myers: A key reason for undefined behavior is to allow the values to be unstable, perhaps due to compiler transformations.
Sewell: I am not hearing any defense of keeping the address-taken distinction. Maybe we can remove it?
Uecker: What is nice about the address-taken approach: We have ways to copy uninitialized objects through things like memcpy(), which is useful. These always require addresses.
Sewell: I see the same consistency.
Keaton: The reason for the address-taken wording was a committee compromise: support for the Nat flag. There is no other fundamental reason for the address-taken.
Sewell: OK. Given the timing, how committed should we be to the NaT flag, since only one architecture did rely on it?
Svoboda: What was the NaT flag?
Keaton: Reference from *Myers*: http://www.open-std.org/jtc1/sc22/wg14/www/docs/dr_338.htm was the issue with IA64 NaT bits.
Sewell: Last question before the break: Should we make using uninitialized values undefined behavior across the board?
Krause: Use cases of a software tool using hidden bits are widespread (e.g.: valgrind, UBSan)
Uecker: I disagree with the last point on load vs. operating on uninitialized values. The error is in the load.
Sewell: It would be useful for a partially-initialized struct to yield useful initial values (while others are uninitialized). We need reads of character types (for memcpy()). Should we allow reads of non-character variables of types that have no traps?
Bachmann: We should allow copying characters because it is more efficient.
Weidijk: What is the difference between an initialized struct and a struct with initialized values? Is a value initialized if I write only some of the bits?
Svoboda: ISO C likens a struct to an array of chars, so reading a struct as array of chars is not undefined behavior. Reading as other types might be (depending on the type). Likewise, C++ constructors can throw exceptions, reading objects can have parse errors. So in my opinion, we cannot allow non-character reads, although specific implementations can allow it on types with no trap representations.
Sewell: Is it useful to allow non-struct non-character type reads of uninitialized variables?
Uecker: Non-character non-struct types should allow undefined behavior on reading.
Sewell: Should we make uninitialized reads of non-struct non-character as having defined behavior always? (3)
Sewell: Should we make uninitialized reads of non-struct non-character as having undefined behavior always? (10)
Sewell: Interesting. Can anyone speak about C++ for this?
Ballman: I do not feel comfortable discussing this. Perhaps you can ask Richard Smith or Jens Maurer.
Sewell: I would like to hear from those who prefer the always-defined approach.
Ballman: I am not convinced by the "undefining" arguments. I prefer 'unspecified value'.
Sewell: Perhaps we should vote about making uninitialized reads of types that do not have trap representations as either being a compile-time error or unspecified value?
Bhakta: What are we voting on?
Gilding: I would prefer to allow that because of the valgrind case, and that is not enumerated by either of the options.

Sewell: For types that do not have trap representations, should we allow non-character types non-undefined behavior reads of uninitialized variables? For now consider only plain scalar variables.

1. always defined behavior, as an unspecified value (3)
2. at implementation's per-instance choice, either compile-time error or unspecified values (3)

3. always undefined behavior (12)
4. other (0)

If we choose 'always undefined behavior' we still need wobbly values for uninitialized character-type reads..

Seacord: I once had a discussion with Clark Nelson from Intel. Intel is fine with eliminating trap representations; we make them all undefined behavior.

Uecker: We should not eliminate trap representations. The question is when undefined behavior happens?

Goldblatt: My vote on (3) is mainly about dynamic checkers (e.g.: valgrind). I would be interested in specifying what happens after an uninitialized value is read, because these checkers would need to handle those.

Svoboda: I prefer having trap representations. As indicated by Annex L, undefined behavior is a big monster that needs to be partitioned and tamed. So I prefer (2), it is the same for checkers as (3) unless the checker can infer your platform restrictions.

Ballman: Runtime undefined behavior for everyone is a bigger problem; we want to deal with that without making it always undefined behavior. We may just need a better term (not "unspecified value" or "undefined behavior" or …)

Uecker: I prefer (2) because of the valgrind issue. In the past, some platforms use aggressive optimization, but that is a quality of implementation issue.

Sewell: Contracts (a la C++) are a massive additional feature.

Keaton: Annex L offered something between (1) and (2). It is conditionally normative. I still like Annex L but no implementation has followed it. So people still are leaning towards undefined behavior, partially for optimization.

Sewell: Undefined behavior is certainly easiest.

Keaton: No customers have demanded Annex L sufficiently strongly.

Gustedt: Having an uninitialized scalar variable is indistinguishable from uninitialized memory in my opinion,. Reading memory you get from malloc() should be undefined behavior.

Sewell: The abstract machine should make regions from malloc uninitialized.

Svoboda: The Sun tarball vulnerability caused uninitialized memory from malloc to be written to tarballs, when its contents was actually the *etc/passwd file (back in the day when /etc/passwd actually stored encrypted passwords). Sun fixed this by using calloc(), which zeroes newly-allocated memory, instead of malloc().*

/*Sewell*: That depends is whether an abstract machine considers values uninitialized.

Svoboda: Annex L is for developers, perhaps we need to sell it more (as a security tool)?

Sewell: If I was confident we wanted (3), we could do a paper based on that, but I am not.

Bhakta: The market has spoken in what it wants.

Sewell: I wonder what your customers would say if you offered them the "no undefined behavior" option?

Bhakta: Most compilers have minimize-undefined-behavior options.

Svoboda: I would like to see our vote on this poll in October. People are still thinking about it here.

Uecker: Why do people hate undefined behavior?

Sewell: It is clear that undefined behavior is a huge problem. Not handling it is an abdication on our part.

Ojeda: I always wanted a compiler flag to "initialize everything to 0".

Ballman: Undefined behavior is bad. Not because compilers take undefined behavior as a license, but because optimizers assume undefined behavior cannot happen and consequently screw up code. I do not think uninitialized values should be that big of a foot-gun.

Keaton: I tinkered with zero-initialization in LLVM, but internally it is possible to zero-initialize everything. In some cases it sped things up.

Ojeda: I now want the ability for a compiler to mark some variables as uninitialized, or with some value that we can use to indicate that memory is uninitialized.

Uecker: That is what Clang's Undefined Behavior Sanitizer (UBSan) does. If we remove undefined behavior, we should take care that these objectives can be met for different people.

Bhakta: Our users should not be nannied. It is more wrong for us to tell our customers that we know better.
Gustedt: There is a C++ effort to have variables initialized to 0, with an attribute to avoid initialization for specific variables. I have added such an attribute to the core I have written. J.F. Bastien has made tests on that.
Ballman: GCC & Clang have command-line options that allows initializing all autovar-init functions, with an attribute listing exceptions. With regard to trusting the programmer, that is no longer part of our charter mandate. We should not get in the programmer's way.
Seacord: Two classes of vulnerabilities: One is code being optimized out; the other is encryption algorithms which treated uninitialized values as a form of entropy. Optimizing that out weakened the encryption significantly.
Krause: I dislike requiring an attribute for code that use to work but breaks now.
Sewell: I would like to re-ask our previous poll question, but with a bit of tweaking.
Svoboda: I would like (2) but documented (so it is implementation-defined behavior).
Sewell: The platform cannot say what happens in every case.
Meneide: "Implementation-defined" means the implementation documents its strategy, not what happens in every case.

Sewell: Poll: For types that do not have trap representations, should we allow non-character reads of uninitialized variables? For now consider only plain scalar variables.

1. always defined behavior, as an unspecified value ()
2. at implementation's per-instance choice, either a compile-time error or a non-silent runtime error or an unspecified values (8)
3. always undefined behavior (10)
4. other (0)

Thursday

- 5.11 Gustedt, Improve type generic programming [N 2638]
- 5.12 Gustedt, type inference for variable definitions and function returns [N 2632]
- 5.13 Gustedt, Simple lambdas [N 2633]

Gilding: These are all implementable.
Krause: Languages are getting more complex. It will be harder to implement languages in the future due to growing complexity.
Gustedt: The implementation burden should be less than you think. In all my proposals here (outside type-generic lambdas), one can re-copy code from working implementations.
Hoffner: What is the overall driving goal? Do we want type-safe C or features from other languages?
Gustedt: We need to program in a type-generic & type-safe way that is better than what we currently have.
Hoffner: As a C programmer I need to be able to read all C code (even if I cannot write it).
Bachmann: There are not that many implementors, but there are many C users. Many are hardware (not software) developers. Saying "you do not have to use this feature!" is not convincing, because I have to understand what I read. I am concerned with adding new features in general.
Gustedt: We should not add features just for the fun of it. But I see needs for these features. I spoke with a C++ programmer who is avoiding C because they need lambdas.
Myers: With regard to complexity, "auto" is straightforward, but lambda is not. They prevent a compiler from parsing C in one pass. Type-genericity gets very complicated.
Gustedt: Agreed; that is why I made "auto" a separate feature.
Ballman: The implementation burden argument needs demonstration. Small compiler shops should not hinder progress of C. In my opinion, lambdas are transformative for C.
Gilding: We found implementing lambdas to be not as complicated as it sounds. It can be done by one person

over several weeks, not months. It is a syntactic transformation. Without the type-generic part, it is single-pass. The complexity comes from interactions with other extensions like GCC's goto out of a statement expression. This will be transformative to the C community as a whole. The rest of the world has moved to callback-oriented programming due to Javascript; this might make them interested in C again.
Seacord: The defer TS would also benefit greatly from lambdas.
Gustedt: Defer would need the lvalue stuff from my 4th paper.
Bhakta: Being transformative is not necessarily a good thing. We may get new users although I doubt that. We do risk alienating the users who stay on C because it is easy to understand.
Ballman: We do want to make sure code is readable. C today is not. Any new syntax has the same problem. We combat that by using pre-existing implementation experience. I appreciate that Gustedt's examples are based on experience from other languages.
Wiedijk: What should be my mental image of lambdas? or variable captures?
Gustedt: Here is a simple implementation analogue: Lambdas look like a normal function, with a block of data on the stack that are captured values.
Wiedijk: So if I put a lambda inside a loop what do I get?
Gustedt: You get a different (lambda) function for each iteration in the loop and it disappears when the loop exits.
Gilding: If sizeof(lambda) was defined, it would help a user develop a mental model of a lambda.
Gustedt: There is no sizeof here, also no copying or address of a lambda. I wanted to make no applications for ABI's; it does not have to be consistent between even different versions of the same compiler.
Bhakta: I appreciate trying not to break ABI.
Svoboda: Without consistency between compiler versions, you have re-introduced ABI in-compatibility.
Gustedt: No, lambdas are internal to C functions.
Svoboda: But could I write a bsearch() replacement that uses lambdas?
Gustedt: No, this definition of lambdas forbids that. Your headers cannot have lambdas; they can only be used by function bodies.
Straw Poll: Does the committee wish something along the lines of an auto feature based on N 2632 in C23?
13-6-2 clear direction
Myers: The details of the "auto" feature are less clear.
Gustedt: If at some point people do not like auto, we can take them out.
Gustedt: This is still subject to future stuff, right? (Right!)
Banham: How do these lambdas compare with C++ lambdas?
Gustedt: For now there would not be lvalue-captures (which C++ has). Also there would not be auto captures.
Straw Poll: Does the committee wish something along the lines of a lambda feature based on N 2633 in C23?
10-5-5 clear direction to continue
Gustedt: The slides for the latest document (N 2675) are online here: https://hal.inria.fr/hal-03165732/document
Bhakta: I did want more discussion on the auto side.

- 5.14 Uecker, nested functions [N 2661] (1 hour)

Uecker: Slides are available here: http://wwwuser.gwdg.de/~muecker1/nested-functions_2021-03-11.pdf
Bhakta: Regarding alternative 3 (trampoline ptr): Where are function arguments handled?
Uecker: Yes, I did not add that to the slides.
Wiedijk: Why does the data need to be on the stack?
Uecker: The frame need not be on the stack. You could pre-allocate the code.
Wunsch: There are machines that can only execute code on read-only memory.
Banham: 3 different effects: (1) nested functions, (2) using goto to exit a nested scope, and (3) function templating.
Uecker: With regard to control flow, we could disallow it.
Gilding: I would say being the wild is what kills the feature. If you adopt GCC syntax, you get GCC semantics. Particularly capturing lvalues, via fat pointers. This does not work across all platforms that GCC

supports. It would be rude to adopt GCC syntax with distinct semantics. We need the hat-pointer syntax. Lvalue-capture is very C-like. It is consistent, but not with positive aspects of the language. Forward declaration syntax is very weird, not discussed here.
Uecker: I like lvalue capture because it is easy to understand. With regard to compatibility, I do not see the issue. GCC could adapt our syntax.
Bhakta: Was there any consideration given to disallowing captures now, and adding them in later?
Uecker: Yes, but that would be very limiting. It would not break things if we add our own pointer type.
Gustedt: Nested functions vs. lambdas would have default lvalue capture, assigned to a local auto variable. Nested functions can also do recursion. Uecker also added the goto feature, which I dislike.
Ballman: I am confused about C++'s std::function; it is meant to be a type that allows you to call member functions, lambdas, and nested functions generically. It is not used to define functions. Clang does not support nested functions because of an explicit decision made in 2007. There were too many security concerns with the GCC syntax & semantics. It is unclear if LLVM could even support GCC syntax or semantics.
Uecker: Was the executable stack a problem?
Ballman: The problem was more semantic, but I know few details.
Uecker: Lambdas have a subset of nested functions semantics.
Gustedt: Recursions for nested functions will require more complexity.
Straw Poll: Does the committee wish something along the lines of N 2661 in C23? 5-9-7 Committee does not want to go this direction.

- 5.15 Krause, @ in basic source character set [N 2639]

  Gilding: This is intended to have no impact on what is allowed for identifiers, right? (Right!)
  Banham: Why not also support the execution character set?
  Krause: My use case was characters in comments.
  Banham: It would be difficult if the execution character set is different than the basic character set.
  Krause: We can look into that.
  Bachmann: We should not allow "$" in identifiers because many tools use "$$" in identifiers.
  Ballman: One use case was "@" inside string literals. This might be something for the C++ compatibility group to study further.
  Straw Poll: Does the committee wish to require "@" in the source character set in C23? 13-0-9 clear direction to proceed
  Straw Poll: Does the committee wish to require "
  " $inthesourcecharactersetinC23?11 - 0 - 11cleardirectiontoproceed/Bhakta/ : Thesearenotbindingsince$
  $. Iamdisinclinedtovoteyesbecause Idonotseeexactwordingyet. /StrawPoll/ : Doesthecommitteewishtorequir$
  in the execution character set in C23? 7-0-15 clear direction to proceed

- Revisit: 5.5 Seacord, Specific-width length modifier (updated from Tuesday) [N 2680]

  Straw Poll: Does the committee wish to adopt everything except "wfn" from N 2680 into C23? 17-0-3 So it goes in.
  Straw Poll: Does the committee wish to adopt "wfn" from N 2680 into C23? 13-3-4 So it goes in.
  Bhakta: For those who voted "yes", was the intent to remove the pre macros with the formatted string?
  Seacord: We have not proposed to remove the existing macros.
  Ballman: I loathe the existing macros. But I do not see a reason to remove them. I thought it would be inconsistent to omit "wfn".
  Bachmann: We have to keep the old macros for compatibility. It is not nice to have lots of duplicated functionality.
  Wunsch: Existing macros are clumsy to use.

Friday

Hedquist: WG21 meetings will be virtual throughout this year

- 5.16 Tydeman, Missing DEC_EVAL_METHOD [N 2640]

  Straw Poll: Does the committee wish to adopt N 2640 as is into C23? 18-0-1 this goes in

- 5.17 Tydeman, Missing +(x) in table [N 2641]

  Straw Poll: Does the committee wish to adopt N 2641 as is into C23? 16-0-1 this goes in

- 5.18 Tydeman, Quantum exponent of NaN [N 2642]

  Myers: You say cases where the formula is undefined by that rule. So does 1^Infinity = 1 when quantum exponent is 0?
  Tydeman: That is the POW(1,Infinity) case? (Yes)
  The formula says you take the floor(Infinity) which would be Infinity. But you cannot have a quantum exponent of Infinity.
  Myers: It would be -Infinity.
  Tydeman: IEEE 754 has a rule that the default quantum exponent is 0.
  Myers: It could be Infinity * 0 which is undefined, but it could be something else. Which rule takes precedence?
  Tydeman: No one in IEEE 754 raised that issue. I can solicit their opinion.
  Ballman: What are the chances that IEEE would come up with distinct wording from us? Can we leave this underspecified and wait for their response?
  Tydeman: IEEE 754 deferred to CFP group when CFP group was ahead of IEEE 754. So they would probably follow our lead.
  Action Item: *Tydeman*: Will bring this issue (POW(1, -Infinity)) to IEEE 754 and get their opinion. (N 2642]

- 5.19 Tydeman, Negative [N 2643]

  Svoboda: Does IEEE 754 or some other more mathematical standard define "negative" and does it include negative 0?
  Tydeman: I do not know.
  Gustedt: In the C standard, a lot of functions already suggest "less than 0" rather than "negative". However "negative" is still used sometimes.
  Keaton: ISO 2382 does not define "negative". So we can define it however we like, as long as we are consistent.
  Bhakta: IEEE 754 does not define "negative".
  Keaton: Since the C standard now uses two's-complement integers, negative 0 is no longer an issue for integers, it is only an issue for floating-point types.
  Tydeman: And N 2643 addresses the floating-point issues.
  Straw Poll: Does the committee wish to adopt N 2643 as is into C23? 16-1-4 it goes in
  Action Item: *Tydeman*: Will search for other uses of the term 'negative' in the draft standard. [N 2643]
  Bhakta: I voted no because there are probably implementations that consider -0 to be negative. So this might break those implementations. I am representing CFP on this discussion point, so I did not feel comfortable bringing this up during discussion.

- 5.7 Svoboda, Towards Integer Safety (updated from Tuesday) [N 2683]

  Bhakta: Process issue: We originally deferred to look at the revision N 2669, now we have another revision of the revision made during the meeting. I was previously willing to compromise on this, but am not willing to take binding votes against this update.

  Svoboda: The main changes in this update:

- Normative text wording only, no "deep" semantic changes. The API remains the same.
- Added recommended practice to all macros.

Core proposal changes:

- Changes only to the normative text
- The constraint changes to name the types rather than the objects
- *result is named as the output explicitly
- Various minor wordsmithing
- Recommended practice added to emit a diagnostic if the types are not correct

Supplementary proposal changes:

- Recommended practice added to emit a diagnostic if the user tries to query checked-integer information for some non-checked-integer type
- Similar recommended practices on all the other macros
- The wording has been made more standard

Gustedt: Process issue: Can we vote quickly on this at the next session?
Keaton: What does this mean?
Gustedt: Wordsmithing is done and the committee is in favor, so could we collect objections before the next meeting and schedule time for a vote only?
Ballman: Traditionally we take a binding vote with a delay to allow for objections to be raised before the next meeting.
Keaton: Especially because Svoboda will not be at the next meeting - is this an acceptable compromise?
Bhakta: It is harder to stop a process that has started than to prevent it from starting. I agree with the idea, but going too far to allow voting in the in-meeting changes. We are not actually likely to re-read a paper after the meeting, which makes it too easy to miss problems. I dislike this compromise and prefer Gustedt's suggestion because the paper review would be before the meeting.
Hedquist: I support Rajan's position.
Gustedt: Even without Svoboda, as long as we discuss this on the reflector, someone else can champion the paper.
Myers: I would like to suggest separate votes on the Core and Supplementary proposals.
Svoboda: I was going to propose that anyway.
Keaton: So we discuss this on the reflector, and then have two votes at the next meeting.
Svoboda: I am interested in other questions on the proposal itself?
Bhakta: I am OK with directional votes, just not with voting this wording into C23 as is today.
Svoboda: We can have a non-binding vote, but that seems unnecessary.
Keaton: We defer this until the next meeting, So we are done for now.

- 5.9 Ojeda, secure_clear [N 2682]

Svoboda: How can a platform know that it should "clear other copies" or "prevent them from being made" before memset_explicit() is called? This is a property of the data, not of the call. That is too hard for an implementation.
Ojeda: That is why that sentence uses "might". A compiler might do this, but most will not.
Wiedijk: If I copy that data myself in memory, is this function allowed to erase that as well?
Hedquist: Should the word "might" in the footnote be "may"?
Ojeda: I do not know.
Hedquist: "might" is not defined, so ISO would have a problem.
Keaton: "May" is a normative word, so it cannot go into a footnote.
Hedquist: So does this material belong in a footnote?
Keaton: The footnote cannot say "may". The ISO says "do not use 'might' instead of 'may'".

Hedquist: You will lose that battle with ISO.
Voutilainen: I am sure there is a way to formulate this within the footnote. Yes, this function could clear other copies made by the user. I am not sure whether that is actually a problem.
Gilding: "might" is used in footnote 115, and 131 uses "can".
Wunsch: "If an implementation is allowed to clear other copies." is outside the footnote.
Svoboda: I do not want this to turn into a "clear all secrets". Either remove this sentence from the footnote, or elevate it to normative text (and submit a separate paper permitting this).
Ojeda: How would an implementation support this?
Svoboda: I do not know. But if an implementation claims to support this, that changes how I would use this function.
Voutilainen: This function could not clear secrets in observable memory, but it could clear secrets in non-observable memory
Gilding: There are intentional other copies and unintentional other copies. It is probably worthwhile to separate out the unintentional other copies.
Voutilainen: The problem is that we cannot demand anything. This function is a dead store. We do not need to explain to implementors how to implement this function. I would hesitate to over-specify this.
Myers: I do not think memset_explicit() will create any copies. If data is in registers, it could be copied to the stack.
Gilding: Alternative 2 does not mandate that "c" is ever used, since "c" is only in Recommended practice.
Voutilainen: There are no guarantees here that an implementation might use "c".
Ballman: Do we need to add to the implementation-defined alternative any constraint that clarifies what happens if you pass a null pointer and nonzero 'n'?
Uecker: Requiring an implementation to do this part would mean this does not have to be a dead store.
Bachmann: The library section indicates that it is invalid to give null pointers to functions unless otherwise specified.
Bhakta: I like Ojeda's suggested wording to Alternative 2: "It calls memset(), then does something implementation-defined".
Bachmann: Can I vote for several choices?
Keaton: We can make this a preference poll, where you can vote for as many as you wish.

Straw Poll: Preference Poll: Does the committee favor intent, as in Alternative 1, or implementation-defined, as in Alternative 2 or calls-memset-and-implementation-defined as Ojeda suggested, in N 2682?
Alternative 1: 15
alt 2: 7
calls-memset-and-implementation-defined: 14
Unclear, but people do not want Alternative 2 by itself.

Straw Poll: Does the committee favor intent, as in Alternative 1, or calls-memset-and-implementation-defined as Ojeda suggested, in N 2682?
Alternative 1: 10
calls-memset-and-implementation-defined: 9
Still not clear.

Voutilainen: The end result will still be implementation-defined.
Keaton: …but without the documentation requirement of 'implementation-defined'.
Straw Poll: Does the committee favor something along the lines of "The implementation might clear other copies of the data (e.g.: intermediate values, stack frame, cache lines, spilled registers, swapped out pages, etc.) or it might avoid their creation (e.g.: reducing copies, locking/pinning pages, etc.)." in N 2682? 8-7-6 no clear consensus
Ojeda: Implementations will not do this if we do not put in that sentence.
Wiedijk: You could phrase it in terms of intention.
Svoboda: Gilding's idea of unintentional copies is good. Also this would be very different as a clear-all-copies function.

Meneide: My preference is that we do Bhakta's suggested Option 3, with a slightly tweaked footnote. I think the footnote is valuable. I can send an e-mail to the reflector.
Voutilainen: I expect implementations to employ anti-SPECTRE mitigations on these techniques. Whether it is said in this footnote or not.
Ojeda: I am not sure if we want this to happen, it might slow the function, or become non-portable.
Bhakta: At one point, we suggested an attribute like [nodiscard]…what happened to that? (e.g.: an 'always-make-this-function-call' attribute that you could apply to memset)
Ojeda: We did bring up attributes, I do not remember what came of that.
Ballman: The C++ Liaison group's April agenda is full, but we could discuss this in May, in time for the June meeting.

Keaton: Two things we take into account for standard language:
ISO/IEC Directives Part 2: https://isotc.iso.org/livelink/livelink?func=ll&objId=4230456&objAction=browse&sort=subtype
ISO House Style: https://www.iso.org/ISO-house-style.html
House Style says "do not use 'might' anywhere". We use it in a footnote. JTC1 is fighting this. In the meantime, let us avoid "might" when we can.

- 5.13 Gustedt, Simple lambdas [N 2675]
- 5.12 Gustedt, type inference for variable definitions and function returns [N 2674]

Krause: What is the value for the programmer for lvalue capture?
Gustedt: For most use cases, yes (modulo registers) One advantage of lvalue captures: You can access local variables at compile time.
Uecker: The type of a new constant is the same as what you get with "auto", right? (Right!). I could not pass a lambda to qsort() without converting the lambda to some kind of function pointer, right?
Ballman: What does "static variables without linkage without variably modified type" mean?
Gustedt: Size info is taken at runtime.
Svoboda: If there is no "&" operator, then how do I get a function pointer from a lambda?
Gustedt: You cast your lambda to a function pointer and can then take address.
Ballman: Do you intend to make "()" lists mean "no info on function arguments' or "empty list" (like C++ does)?
Gustedt: It is defined to be the same as C++.
Gilding: Do we need these prohibitions (on "=" or "&" for lambdas) now when we do not have hat-pointers?
Gustedt: We will have to lift those prohibitions at some point.
Ballman: I understand not wanting to do trailing-return syntax; that is orthogonal to this proposal. With regard to auto return type, if there are multiple return statements, which is used for the lambda's return type? I suggest we follow C++ semantics in that all return types must be the same. Also, how do qualifiers affect return types? ("const" is not important, but "_Atomic" is.)
Gustedt: Agreed.
Myers: Lambdas increase the number of required tokens lookahead. You need to look two tokens past an "[" to distinguish lambdas from arrays.
Gustedt: …or attributes. A local variable with the same name as an attribute already requires 2 tokens lookahead.
Myers: I do not think a lambda can be used in a constant expression like a designator.
Gustedt: But we have VLAs.
Banham: I am concerned with dropping the return type, it suspends the ability for the compiler to check the return type as part of a function's signature.
Gustedt: In this proposal when you convert a lambda to a function pointer, you have to specify return type.
Ballman: The primary use case for specifying return types is when your lambda is more complicated, perhaps with multiple returns of varying types (such as short vs. int).
Gilding: One other use of C++'s trailing return type is for returning references, which C does not support.
Uecker: It could be useful for returning VLAs.

Gustedt: Maybe. But that should warrant a different paper.
Meneide: Regarding not having a "&": One of my use cases is to use lambdas to create closures, and then use typeof to create a pointer to a lambda and make a function pointer out of that and use it as a callback.
Gustedt: OK, let us make the "&" operator undefined…so you could define it. I do not want to constrain implementations to do it exactly as C++ does.

6. Clarification Requests

The previous queue of clarification requests has been processed.

7. Other Business

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

7.1 Gustedt, type-generic lambdas [N 2634]

7.2 Gustedt, lvalue closures [N 2635]

7.4 Ballman, Adding a Fundamental Type for N-bit integers (updates N 2590) [N 2646]

7.5 Gustedt, Add new optional time bases v4 [N 2647]

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

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

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

7.9 Gustedt, Revise spelling of keywords v5 [N 2654]

7.10 Gustedt, Make false and true first-class language features v4 [N 2655]

7.11 Múgica, Outer [N 2657]

7.12 Ojeda, Safety attributes for "c" [N 2659]

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

7.14 Uecker, maybe_unused attribute for labels [N 2662]

7.15 Uecker, life time, blocks, and labels [N 2663]

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

8. Resolutions and Decisions reached

8.1 Review of Decisions Reached

Does the committee wish to adopt something along the lines of N 2619 into C23? 18-0-0
Does the committee wish to use a "_Typeof" keyword with the usual header for the typeof feature in N 2619? 7-7-5
Does the committee wish to use a "typeof" keyword for the typeof feature in N 2619? 16-2-1
Does the committee wish to use a completely new keyword (rather than typeof or _Typeof) for the typeof feature in N 2619? 1-14-3 Clear preference.
Does the committee wish typeof to accept type names (in addition to expressions) as a valid argument in N 2619? 17-1-4
Does the committee wish remove_quals to accept expressions (in addition to type names) as a valid argument in N 2619? 11-2-5
Does the committee wish to adopt N 2645 into C23? 15-1-4 This goes in to C23.
Would the committee wish to adopt something along the lines of N 2621 into C23? 7-3-8
Does the committee wish to adopt N 2626 with editorial changes into C23? 17-1-0 it goes in
Does the committee wish to adopt N 2630, without recommended practice, into C23? 13-0-1 passes
Does the committee wish to adopt the "recommended practice" in N 2630 into C23? 13-2-3 passes
Does the committee wish to replace the "c" parameter with a specific value, from memset_explicit() in N 2631? 4-8-11
Does the committee wish to replace the "c" parameter with "implementation-defined values", from memset_explicit() in N 2631? 4-9-8
Does the committee prefer Alternatives 1, 2, or 3 as is in N 2631?
Alternative 1: 13
Alternative 2: 4
Alternative 3: 0
Alternative 1 wins
Does the committee prefer removing the Recommended Practice section from Alternative 1 in N 2631? 15-0-7 passes
Does the committee prefer adding memset_explicit() to the exception list in <string.h> freestanding implementations? 3-8-10 So memset_explicit() is not added to the exception list and will be required in freestanding implementations.
Does the committee wish to use N 2577 as the base document for TS 6010? 20-0-2
Does the committee wish something along the lines of an auto feature based on N 2632 in C23? 13-6-2 clear direction
Does the committee wish something along the lines of a lambda feature based on N 2633 in C23? 10-5-5 clear direction to continue
Does the committee wish something along the lines of N 2661 in C23? 5-9-7 Committee does not want to go this direction.
Does the committee wish to require "@" in the source character set in C23? 13-0-9 clear direction to proceed
Does the committee wish to require "
" *inthesourcecharactersetinC*23?11 − 0 − 11*cleardirectiontoproceedDoesthecommitteewishtorequire* " @ "
in the execution character set in C23? 7-0-15 clear direction to proceed
Does the committee wish to adopt everything except "wfn" from N 2680 into C23? 17-0-3 So it goes in.
Does the committee wish to adopt "wfn" from N 2680 into C23? 13-3-4 So it goes in.
Does the committee wish to adopt N 2640 as is into C23? 18-0-1 this goes in
Does the committee wish to adopt N 2641 as is into C23? 16-0-1 this goes in
Does the committee wish to adopt N 2643 as is into C23? 16-1-4 it goes in
Preference Poll: Does the committee favor intent, as in Alternative 1, or implementation-defined, as in Alternative 2 or calls-memset-and-implementation-defined as Ojeda suggested, in N 2682?
Alternative 1: 15
alt 2: 7
calls-memset-and-implementation-defined: 14
Does the committee favor intent, as in Alternative 1, or calls-memset-and-implementation-defined as Ojeda suggested, in N 2682?

Alternative 1: 10
calls-memset-and-implementation-defined: 9
Still not clear.
Does the committee favor something along the lines of "The implementation might clear other copies of the data (e.g., intermediate values, stack frame, cache lines, spilled registers, swapped out pages, etc.) or it might avoid their creation (e.g., reducing copies, locking/pinning pages, etc.)." in N 2682? 8-7-6 no clear consensus

8.2 Review of Action Items

Tydeman: Will bring this issue (POW(1, -Infinity)) to IEEE 754 and get their opinion. (N 2642]
Tydeman: Will search for other uses of the term 'negative' in the draft standard. [N 2643]

10. Thanks to Host

10.1 Thanks and apologies to Jens Gustedt, the originally intended host

10.2 Thanks to ISO for supplying Zoom capabilities

11. Adjournment (PL22.11 motion)

PL22.11 motion by Tydeman, seconded by Ballman. Objections? (None).
Adjourned on Friday, March 12, 2021 at 17:30 UTC.