Minutes for Aug 25 - 29, 2025

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

WG14 / N3684

Dates and Times

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

25 August, 2024	09:00 – 12:00	Lunch	13:30 – 16:30
26 August, 2024	08:45 – 12:00	Lunch	13:30 – 16:30
27 August, 2024	08:45 – 12:00	Lunch	13:30 – 16:30
28 August, 2024	08:45 – 12:00	Lunch	13:30 – 16:30
29 August, 2024	08:45 – 10:45		

Meeting Location

Faculty of Business Mendel University in Brno
Building Q, Deanery at Ground Floor
Zemědělská 1665
613 00 Brno-sever-Černá Pole
Czechia

Meeting information

Venue information:
N3541: Brno August 2025 Venue

Meeting Scribe: David Svoboda

Local contact information

Hana Dusíková

1. Opening Activities

1.1 Opening Comments (Seacord)

Welcome to the 73rd meeting.
Seacord likes to keep everything on schedule. Leave time for straw polls - at least 5 minutes.
Wait until section of agenda to hear from national bodies.

1.2 Introduction of Participants/Roll Call

Name	Organization	WG14 Country	Notes
Aaron Bachmann	Efkon GmbH	Austria	Austria NB
Aaron Ballman	Intel	USA	
Alejandro Colomar	Linux	Spain	Spain NB
Alex Celeste	Perforce Software	USA	MISRA
Ash Chronister	Apple	USA	
Aymeric Baud	AFNOR	France	
Chris Bazley	Arm	UK	
Clive Pygott	LDRA Technology	USA	
Daniel Plakosh	SEI/CERT/CMU	USA	
Dave Banham	BlackBerry QNX	UK	MISRA Liaison
David Svoboda	SEI/CERT/CMU	USA	Scribe, UB SG Chair
David Tarditi	Apple	USA	
Eskil Steenberg	Quel Solaar	Sweden	Sweden NB
Fred Tydeman	Keaton Consulting	USA	INCITS/C Vice Chair
Freek Wiedijk	Plum Hall	USA	
Glenn Coates	Real Time Embedded Ltd	UK	
Hana Dusíková		Czechia	Host
Henry Kleynhans	Bloomberg	USA	
Jakob Łukasiewicz	Motorola Solutions Systems Polska	Poland	Poland NB
JeanHeyd Meneide	NEN	Netherlands	Neth. NB; C++ Comp. SG
Jens Gustedt	INRIA/ICube	France	France NB
John McCall	Apple	USA	
Joseph Myers	Red Hat	UK	UK NB
Joshua Crammer	Intel	USA	
Martin Uecker	Graz University of Technology	Austria	
Maryam Karampour	Aviar Inc.	Canada	Canada NB
Nevin Liber	Argonne National Laboratory	USA	
Niall Douglas	Ireland ned Productions Ltd	Ireland	Ireland NB
Omar Sandoval	Meta	USA	
Patrizia Kaye	Self	UK	
Paul McKenney	Meta/Facebook	USA	

Name	Organization	WG14 Country	Notes
Pavel Simerda	Self	Czechia	
Philipp Krause	SDCC	Germany	
Rajan Bhakta	IBM	USA, Canada	USA NB; INCITS/C Chair
Robert Seacord	Woven Planet North America Inc	USA	
Thiago Adams	ABNT	Brazil	Brazil NB
Vince Mailhol	Woven Planet North America Inc	USA	
Vlad Srebrennikov	Halpern	USA	
Yair Lenga	Self	Israel	Guest
Yeoul Na	Apple	USA	

1.4 Required Reading

- 1.4.1 ISO Code of Ethics and Conduct
- 1.4.2 ISO Guidance and Process
- 1.4.3 IEC Code of Conduct
- 1.4.4 JTC 1 Summary of Key Points N2613

1.5 Approval of Previous Minutes (Feb 2025) N3583

Bhakta moved. **Ballman** seconded. Objections? *None*

1.6 Review of Action Items and Resolutions

Ballman will update the retcon list with items from the document review (all handled during the session: nothing outstanding).
Ballman: Done

1.7 Approval of Agenda: N3680 on Google Docs

Bhakta moved. **Pygott** seconded. Objections? *None*

1.8 Identify National Bodies Sending Experts

Person	Nation
Eskil Steenberg	Sweden
Hana Dusíková	Czechia
Jakob Łukasiewicz	Poland
JeanHeyd Meneide	Netherlands
Jens Gustedt	France
Joseph Myers	UK

Person	Nation
Martin Uecker	Austria
Niall Douglas	Ireland
Philipp Krause	Germany
Rajan Bhakta	Canada & US
Thiago Adams	Brazil

2. Reports on Liaison and Collaboration Activities

2.1 ISO, IEC, JTC 1, SC 22

Bhakta: The USA INCITS meeting is in two weeks, Monday. Please attend.
Seacord: I attended SC22 meeting in London last week remotely.

2.2 National bodies in WG 14

None

2.3 WG 21

Patricia: We have published a CD for NB comments.
C++26 will be based on C23.
Bazley: I emailed **Nina** and cc'd **Meneide** about his syntax extension for WG21. I have gotten no reply.

2.4 WG 23

None

2.5 MISRA C (straw poll on communique)

Celeste: There is no current interesting progress to report. MISRA is calling it "C24".

2.6 Austin Group

None

2.7 Unicode Consortium

None

2.8 Other Liaison or Collaboration Activities

None

3. Study Groups

3.1 C Floating Point Study Group activity report (Bhakta)

Bhakta: I am continuing to examine floating-point issues from the community, and fixed-point arithmetic, too. CFP are interacting with the IEEE committee about floating-point issues. Jim Thomas is our editor and convener. For issue 1014 (to be discussed under N3610), we need to decide what happens with regard to error handling.

3.2 C Memory Object Model Study Group activity report (Sewell)

Wiedijk: The reflector is not working for me right now.

3.3 C and C++ Compatibility Study Group activity report (Meneide, Nina)

Meneide: There were a couple of divergences in the preprocessor after C++26 was finalized. They have questions about how certain directives are handled, as well as white-spacing issues. Many will be handled by the "nvec-c" paper. There was a general motion on VLA parameters. They are mostly waiting for WG14 to figure things out.

3.4 Undefined Behavior Study Group activity report (Svoboda)

Svoboda: There are several papers published: Earthly Demon papers. the Examples of Undefined Behavior paper, and the Educational Undefined Behavior paper (actually on Github). The latter two should become white papers. We need new work item proposals for both papers. Our next Undefined Behavior SG meeting is Friday, Sep 5.
Uecker: Here is a paper showing our earthly demons effort. It includes a chart of which undefined behaviors have been "slain".

3.5 defer (Meneide) TS (Committee Draft), r2 [N3589]

Bhakta: Has anyone asked if the floating-point TS supports exception handling? Part 5 of the TS has a floating-point exception handling mechanism. Do we want to update the floating-point TS to handle "defer"?
Meneide: Nobody has raised concerns with me about it. We have answered all semantic questions. I would like to go to SC22 for getting out a ballot for a comment period.
Bhakta: Can we add paragraph numbers? It makes it easier to comment.
Meneide: Sure, for the next N-document.
Seacord: We could do an offline vote for submitting.

3.6 Memory Safety (Uecker)

Uecker: The first meeting happened.

3.7 _Optional Study Group (Bazley), Dog-fooding the _Optional qualifier (v2) N3597

Bazley: **Seacord** informed me that I should be setting up a study group. I would like some help with supporting _Optional.

3.8 Infrastructure Group (Meneide)

Ballman: **Srebrennikov** and Derek Jones and I worked on recovering documents. We got 300 old docs from 1994 and earlier; they have been added to the document log. And thanks go to **Plakosh** and Keld Simonsen for backed work. A lot of the old documents are still relevant.
Gustedt: It is simple to set up another mailing list with more modern software. Each SG has their own.

Ballman: Simonsen is waiting for a request for updated mailing-list software. So we will have a simple way to cite messages.
Myers: We still have message numbers for WG14 reflectors.
Meneide: I am working on a privacy policy; currently paused.
Action Item: **Seacord**: Work with Simonsen to update the WG14 reflector.
Myers: There is a new system for assigning document numbers in the WG14 document log. I am also adding references for old meeting minutes for when documents are discussed.

3.9 New Study Group proposal (if applicable)

Bhakta: What happened with the "embed" TS? Is there a SG?
Seacord: You mean the embedded processor. I thought that an SG to update the embedded TR would help.

CRFI Study Group (Celeste)

Celeste: This ties into some things: We should learn from **Myers**' page about C documents. CRFI is a little like Scheme, a request for implementations for list of extensions. You cannot extend C with cool features like you can with Scheme. So there will be a repository for library API descriptions. It should be maximalist, allowing for multiple ways to do the same thing. The strp_t implementation by **Bazley** has already been added. It also has some things from **Colomar**. It has minimal gate-keeping, similar to asking **Plakosh** for document numbers. Compilers can then indicate support for these extensions. So CRFI is a TS for "extended library features".
Bazley: **Colomar** and I have discussed several string proposals. I do not think our proposals are adversarial.
Bhakta: Do you see this in addition to what WG14 does?
Celeste: WG14 has a reasonably flexible process. Perhaps we can relieve pressure on WG14 by providing an alternative repository for library proposals.
Gustedt: Some features still make sense to handle directly in WG14.
Seacord: We need a new work item to publish a TS. I can approve a study group.

4. Future Meetings

4.1 Future Meeting Schedule

Spring, 2026	Part 1, February 2 - 6	Virtual
Spring, 2026	Part 2, March 9 - 13	Virtual
Fall, 2026	September 14-18	301 Moodie Drive, Ottawa ON, K2H 9C4

Candidate Host: Suan Sunandha Rajabhat University
Action Item: **Celeste**: Contact **Bhakta** about the Canada (Fall 2026) meeting.

4.2 Future Deadlines

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

Pre-Spring 2026 Part 1	2 January, 2026
Post-Spring 2026 Part 1	6 March, 2026
Pre-Spring 2026 Part 2	9 February, 2026
Post-Spring 2026 Part 2	13 April, 2026

5. Document Review

Monday, 25 August

- **Celeste**, Straw poll on TS 25007 "C Extensions to Support Generalized Function Calls", committee draft [N3582] (5 minutes)

  Bazley: "Function literals" in this TS it not normative. It may not be perceived that way by people reading it.
  Ballman: For implementers, are you going to implement this TS? Clang will not implement this, but we are not opposed to it either.
  Bhakta: We are unlikely to implement this.
  Krause: At the end of a declaration, you could replace it with attributes.
  Bhakta: I prefer to keep the text as non-normative. I would like to see this TS published.
  Straw Poll: Would WG14 like to submit TS 25007 to SC22 for balloting? 13-0-8 + 0-3-4 = 13-3-11: consensus

Morning break

- **Uecker**, Arrays (0.5 hours)
  - Comments on Array Type [N3428]
  - Matching of Multidimensional Arrays in Generic Selection Expressions (updates N3290) [N3348]

  Celeste: In C11 one could match types that decay using compound literals. But this is convoluted.
  Ballman proposed _Generic over types, which matches **Uecker**'s proposal exactly.
  Bazley: It is important that WG14 decisions should be based on real-world usage. I am Not sure that "more precise" is necessarily "better". But I fully support this paper.
  Bhakta: Is **Meneide**'s paper still in play?
  Mychal: How does this affect extended constant expressions?
  Uecker: The other way around.
  Meneide: I plan on bringing my paper back, but in modified form. It is not done for this meeting. I am not really against **Uecker**'s changes. You do not want to distinguish constant array vs. non-constant array in _Generic. We distinguish the kind of undefined behavior you get from having a compile-time vs. runtime size. This was previously a constraint violation. This paper pushes runtime size array with runtime size, which is only caught by UBSan or something like that. By allowing variably-modified types here, you increase undefined behavior when there is a mismatch of sizes.
  Celeste: I am not convinced this is a new undefined behavior. It is fine to introduce this feature now and fix it later.
  Bazley: I know a lot of people do not like variable-length arrays. They would become useless if they could not be used in place of a statically-sized array.
  Ballman: If I have int[n] type, where n will never be evaluated, how do I match that at compile time? It is still based off the regular type compatibility rules, right?
  Uecker: I did not change the rules for _Generic. It is undefined behavior if array sizes do not match at runtime.
  Meneide: This is why I wanted more sophisticated rules about this. Two arrays are compatible until you detect that their sizes differ.
  Gustedt: Compatibility is a compile-time property. If one array is fixed and the other is a VLA, they can match at compile time. It is unfortunate that the standard text also mentions runtime undefined behavior in the same paragraph. _Generic originally was modeled around function calls. This proposal goes exactly in that direction.
  Bazley: I would like to see an operation that rebuts what **Uecker** has written.
  Ballman: There is a lot of deficiency in this space. If we fix those, it surprises users. Do you share this concern?

Uecker: Yes.
Ballman: Today here is what we need to fix: those cases would still be rejected because you have multiple associations somewhere.
Uecker: If you have runtime sizes, there is no obvious static way to fix it. You can have runtime checking.
Ballman: In the future we could fix this with runtime type checking. For other cases, like static_assert, this would be a constraint violation.
Mychal: If we did this variably-modified restriction, could we restrict variables with a VM descriptor?
Uecker: We could merely add the ∗.
Meneide: This what I was trying to do in my paper. If we accept this, I can rework my paper.
Straw Poll: Would WG14 like to adopt N3348 as is into C2y? 13-6-6 + 3-0-2 = 16-6-8: consensus

- The `__COUNTER__` argument
  - **Gustedt**, The `__COUNTER__` predefined macro [N3457] (0.5 hours)

Celeste: I can think of a way in which requiring a suffix would actually break.
Bazley: If this recommended practice is worthwhile, it should apply to `__LINE__`, which is the best precedent for `__COUNTER__`. I would prefer not to have the suffix.
Gustedt: There is some room for improvement with regard to suffixes. For architectures with 17-bit integers, I have no idea if it is worthwhile to require a suffix or not.
Ballman: The existing practice is that `__COUNTER__` expands to an int. Most implementations are giving an int today.
Gustedt: It uses int when it can represent all values, and logs when it can not.
Myers: The evaluation has an off-by-one error; it implies the largest value is 2^31-2.
Krause: I do not expect users of this to rely on the type.
Straw Poll: Would WG14 like to adopt N3457 into C2y without the recommended practice first line and two bullets, and 2147483648 as the limit? 24-0-2 + 3-0-1 = 27-0-3: consensus

  - **Johnson**, Counter Directive [N3636] (0 hours)
- **Chronister**, Initializing static and thread objects with compound literals [N3235] (0.5 hours)

Bazley: I spent hours reviewing this paper; I have to vote against it. The wording is wrong and the footnote can be removed. I would like to see a revision of this paper. The standard already defines "compound literal constant". I suggest revising the definitions of "named constant" and "compound literal constant".
Myers: I agree with Bazley about the "constant" phrase.

Lunch Break

- **Meneide**, `__self_func`, r2 [N3486] (0.5 hours)

Bhakta: Are there any thoughts to add this to the generalized function calls TS?
Meneide: The TS would only use `__self_func` as an example, not a normative feature.
Celeste: We had a vote on clause 4 of the generalized TS.
Krause: There is no implementation experience; compilers never added this.
Bhakta: That is why I asked about this.
Ballman: Clang does not have a need for this. It is a feature for feature's sake.
Colomar: I could vote for this after we have function literals
Gustedt: I would use this in macro generations.
Meneide: There are three use cases: One is macros. One for va_args; you can use this to determine properties of the function, blocks, and other existing practice of function types.
Bazley: This is not something I have ever wanted and yet I am tempted to vote in favor of it due to its simplicity. If it is introduced, people would use it to be more explicit when doing recursion. But in reality, recursion often goes through more than one function, so it is not as useful without further motivation. Otherwise, I have no further problem with it.
Bhakta: If you do not see the need but it is simple, that is a sign it should not be added. It is

implementation experience that is needed. Once it is implemented, then we know it is going to be used.
McCall: The nested functions and blocks were brought up as a reason to have something like this. We do have a use for that; it is something we have considered but it has not been sufficiently important to get it implemented. It is not obviously the right design though. When you start talking about anonymous functions and self-reference, you have a nesting issue. You can reasonably want to refer to your parent in the same sort of self-reference way.
Colomar: A double underscore "__" usually indicates implementation namespace. Can we respect that separation?
Celeste: This says "primary expression" but uses syntax of a postfix expression.
Meneide: That was a typo on my part; it should be "postfix".
Seacord: Does this mean a revision to the paper?
Meneide: Yes, I will revise it. We should wait for stronger motivation.
Bhakta: Moving this to primary would make it a different paper.
Seacord: I would prefer to see a quick revision and an offline vote.
Uecker: I like the paper, but prefer not to vote for it until we have a use case.

- Forward declarations (0.5 hours)
  - **Na**, Dependent attributes [N3656]

***Ballman**: This is a complicated design space; everyone has opinions. This is so WG14 can endorse this as perhaps a TS so that we can gain implementation experience.
Steenberg: Do we need a new language feature for this?
Na: We are looking for WG14 guidance. If C adds forward declaration mechanisms, that is WG14 indicating how to solve a problem.
Gustedt: We do not have attributes in the standard which would use this pattern. These rules are complicated. We did not intend to specify anything about what goes inside the braces of an attribute.
Celeste: If you have a working implementation, we do not forbid the lookup rules you are suggesting.
Bhakta: I do not see any way to prevent forward declarations.
Ballman: This is not just about attributes. We have constexpr variables.
Bachmann: From the user's pointer of view: top-down, left-to-right. We should not have provided only one way of doing examples.
Na: These attributes will be useful for memory-safe mode.
Ballman: Struct members means we will have multiple ways to handle these.
Bazley: This could easily become a de-facto standard, adversarial to existing syntax, quietly forking the language.
McCall: We are not proposing adding counted_by to the standard. But contracts would imply a new lookup algorithm.
Tarditi: We have code where people declare the pointer first, and length argument second.
Bhakta: For my point regarding late parsing: In your paper, Proposal section, bullet 1: “for all other identifiers, the dependent attribute scope is their ordinary scope.” This does not' seem to restrict any dependent lookup to just members or prototypes. It later, via subsequent bullets, does prescribe the behavior for members/prototypes, but does not seem to prohibit other uses.
Colomar: This is a violation of much of our charter. We need to count array parameters. We need complex expressions for the size.
Bazley: The fundamental premise is wrong. You should be able to write C code in any arbitrary order. If you want to attach invariants to a struct, put the constraints at the end of the struct.
Wiedijk: Perhaps we should move all invariants to the end. When putting function header in memory, we have so much memory nowadays that this is not much of a limitation to developers. can we not just move attributes around?
Ballman: If a function argument location can change the grammar, that will be unexpected. Clang does not have consensus for any approach. We have no consensus on how to do this with attribute; we only have consensus not to support GCC's forward-declared parameters.
Uecker: Who is exactly the Clang community? It does not include me.
Opinion Poll: Regarding N3656, is the committee in favor of proposed-name lookup behavior for dependent attributes? 0-6-1 + 16-4-5 = 16-10-6

Opinion Poll: Regarding N3656, is the committee against diagnosing name conflicts in array size to catch potential errors? 2-1-1 + 1-16-7 = 3-17-8

- **Bazley**, Alternative syntax for forward declaration of parameters [N3433]

Bhakta: We already had a strong opinion poll on this paper in Graz.
Bazley: This is the version that will go into the standard, with mixed blessing from **Uecker** because he wanted the same thing with slightly different syntax.
Bhakta: Can you summarize what the needed changes are? Could you vote this in and then clean it up?
Bazley: It does not explicitly constrain against having un-named parameter declaration. The wording is not tight enough.
Ballman: Putting in something to the standard that we intend to fix later is hostile to implementers. This is something the Clang community has consensus against.
Na: This can be made compatible with our paper.
Opinion Poll: Would WG14 like something along the lines of N3433 to be eventually adopted into C2y? 3-2-0 + 10-12-4 = 13-14-4: no preference

Afternoon break

- Constant Expressions
  - **Gustedt**, Chasing Ghosts I: constant expressions v2 [N3558] (0.5 hours)

Ballman: The term "description" is used with two meanings in the wording; it should be clarified.
Bhakta: Is Clause 1 one of those ISO things we have to do?
Ballman: Yes.
Myers: In reflector message 30480, is this wording you are going to put back?
Gustedt: I do not feel comfortable with floating-point changes.
Bhakta: If I did not get any mail, I did not plan any changes.
Gustedt: This could be done independently later.
Action Item: **Bhakta**: CFP to submit a paper as per reflector message 30480.
Straw Poll: Would WG14 like to adopt N3558 as is into C2y? 4-0-0 + 15-0-6 0 = 19-0-6: consensus

- **Myers**, Open Issue processing C23 issue log, r1 N3609] (1.5 hour
  - Issue 1000 (FAM compatibility)

    Celeste: This does change current actual behavior of GCC and Clang.
    Myers: Right

  - Issue 1002 (type qualifiers in [*])

    Ballman: Our defects list will say "All issues in this list apply to previous versions of C unless stated otherwise."
    Bhakta: I did not see this as "only C23" because of my history with defect reports. For the poll, given what we already have, we can just say "add to C2y".
    Straw Poll: Would WG14 like to accept the suggested correction for issue 1002 from N3609 into C2y and into previous versions of C? 4-0-0 + 14-0-3 = 17-0-3: consensus

  - Issue 1003 (linkage between library functions)

    Bhakta: The link for the C90 issue is broken. Also, no linkage does not mean they cannot be equal pointer values. This is not totally correct, but it is progress, so I am OK with it.
    Straw Poll: Would WG14 like to accept the suggested correction for issue 1003 from N3609 into C2y and previous versions of C? 1-0-2 + 16-0-4 = 17-0-6: consensus
    Bhakta: In this case "matching failure" would work.
    Colomar: For the "like" family, I have similar issues.

Bhakta: Yes, that is issue 1012.
Action Item: **Douglas**: Check with POSIX for direction for issue 1004 in N3609.
Bhakta; Just call it a matching failure. That is our way out.
Douglas: Could this report a compiler diagnostic?
Myers: Yes, if the code gets executed.
Ballman: No implementation is changing any behavior whatsoever, right? This is not a good use of our time.
Colomar: In strtol(), the numbers depend on how much has been parsed. Could we do the same here?
Bhakta: The return clause clearly states what would be returned.
Seacord: This error is a matching failure. It can return the number of items assigned.
Opinion Poll: Is there a third type of failure for issue 1004 in N3609? 2-1-2 + 1-7-9 = 3-8-11

- Issue 1005 (Annex D and UAX#31)

  Seacord: The Unicode Study Group from WG21 had to rebase their standard to the Unicode standard. I recommend this correction; it is a step in the right direction. We probably more work to be done in the future.
  Krause: Partial wording should not be accepted; it should only be applied to incomplete features. We are taking out something, so this is good.
  Straw Poll: Would WG14 like to accept the suggested correction for issue 1005 from N3609 into C2y? 0-0-4 + 12-0-9 = 12-0-13: weak consensus, adopted

Tuesday, 26 August

- **Myers**, Open Issue processing C23 issue log, r1 [N3609] (1.5 hours)
  - Issue 1006 (atomic_fetch operations and "address types")

    Myers: **Gustedt** has a paper which addresses this as well as other things. So we skipped the topic until we get to that paper.

  - Issue 1007 (Implicitly unsigned bit-field ambiguity)

    Myers: There is a weird thing where a bit-field is of type int but it may still be an unsigned type. For C2y, do we want to follow C++14 and say that a bit-field has the same sign as its type? That requires a paper for a future meeting. But we can figure out what we want to do for C23.
    Seacord: The most interesting idea I have heard is to deprecate implicitly signed int to force people to explicitly signed int, just for our bit-fields. That is consistent with the MISRA standard which requires an explicit sign for bit-fields.
    Myers: What counts as being explicitly or implicitly signed? Things like typeof, etc.?
    Seacord: I think my take on the mailing list was that it should be what it is. If it is used as a bit-field, we could deprecate that.
    Celeste: I reiterate that if we force the spelling that will definitely break code for at least some users, so I prefer deprecation. Typeof is not likely to break code for a lot of people. But I do know that people would have silent changes if the behavior changed.
    Gustedt: Typedef int to a bit-field then it is not clear if the field is signed or unsigned, but it is a really weird specification. I do not know how we can get around this.
    Liber: The spelling of the type changes the behavior, but it should not matter how the type is spelled. Typedef should be the same as the underlying type. I think we have to bite the bullet and change this. There was an objection in C++ from SUN because of their headers.
    Bazley: I do not see the point in deprecating implicitly unsigned or implementation-defined unless a diagnostic is issued. So I am inclined to just fix it. The people who need to change their code would not know they need to update their code.
    Bachmann: In the long run "int" should be "signed int" for consistency.
    McCall: I think the C++ committee's philosophy on this was that there were not too many targets to

do this, so non-conformance is not an issue. That may not match with WG14's philosophy.
Liber: This was changed in C++11
Ballman: Deprecation would be too noisy for implementations in practice; int is a very common bit-field type.
Krause: I would still be in favor of allowing this to be unsigned because of efficiency. But with typeof and BitInt issues, it does not feel like we can consistently argue for allowing it to be unsigned. I do not know if this impacts our users. I would suggest that plain int is signed but I would wait to hear if implementers are concerned.
Myers: The poll will be direction so we can write a paper.
Wiedijk: I would like int to be signed, so if the number of bits allows it we could allow the value to be unsigned while the type is signed
Seacord: The other thing from deprecation is silently changing behavior, which is not a conservative approach. Usually we deprecate this and then make a change in behavior.
Liber: But this is implementation-defined, which is a bit different.
Krause: We have these typeof cases and these _BitInt cases, so how do we know if it is an int or not?
Seacord: Where there is ambiguity, I would say that is intended to be signed but make it deprecated for where it is unambiguous
Bhakta: Clang has an option to select signed vs. unsigned. My preference is to make no change. But I think the next step would be recommended practice, then deprecation, etc. A slow progression so we do not break code. I do not expect people to change their code quickly.
McCall: Do we have any census of what platforms rely on this?
Seacord: We keep hearing about Solaris.
Krause: SDCC has unsigned and we do not even given an option to make ints signed, but I do not know how much our users rely on this.
Bazley: **Krause**, do you supported signed bit-fields at all?
Krause: Only if you say "signed int" explicitly.
Bachmann: Bit-fields are used for packing data, so sign and unsigned uses are important.
Seacord: What is the room preference? Leave it the way it is, deprecate, diagnose, or silent behavior change?
Celeste: Is there a meaningful distinction between diagnose and deprecate?
McCall: Diagnose would be "make it a constraint violation".
Gustedt: Diagnose only on targets where int would be unsigned.
Bhakta: I am not clear on whether we are answering the questions like, "What if it is typedef?" That is the crux of a lot of questions.
Myers: We are figuring out what we want to do for C2y and then we can figure out what to do with C23. The answers would most likely not be the same for C23 and C2y.
Opinion Poll: Would WG14 like to leave as-is in Issue 1007 in N3609? 10-11-2
Bazley: Is not this about portability? Whose problem are we trying to solve?
Celeste: The constraint violation only manifests on implementations which already are not going to want to change their behavior because it breaks code?
McCall: Implementations become non-conforming.
Celeste: Silently.
Krause: The problem we are solving is the confusion among users, particularly with typedef. It is a signed type in some places, but they use it as a bit-field and then it acts unsigned.
McCall: Do we know whether we know what existing implementations actually do for typedefs?
Krause: In SDCC it depends on whether the keyword "signed" is in the typedef.
Opinion Poll: Would WG14 like to make a constraint violation in Issue 1007 in N3609? 9-11-5
Opinion Poll: Would WG14 like to make a silent behavior change to match C++ in Issue 1007 in N3609? 9-9-7
Seacord: No consensus, so we are at status quo which means the first option wins despite the vote.
Myers: Do we want to take votes for what to do in C23?
Seacord: Do we know what our direction is now? I suppose it is to leave it unchanged.
Myers: I am not clear what the direction is for C2y but C23 can be different.

Bhakta: I believe whoever asked the questions is looking for clarity on C23, regardless of C2y.
Bazley: Why don't we just ask **Krause** what SDCC does, because the answers to these questions may be divorced from reality?
Bhakta: No offense to **Krause**, but there is plenty of other implementers.
Liber: Typeof(1 + 1) is not spelled "int" so it is unclear what this means.
Seacord: Maybe we need another paper?
Bhakta: That is why I asked where this came from. Was it us navel gazing or did a user ask for it?
Myers: It is one of the many issues I accumulated since 2017.
Gustedt: we added typeof in C23 so we made the problem worse because that has no answer at all.
McCall: Maybe there is an implementation consensus on typedef? Everyone seems to look through typedefs, at least from the implementations we know of.
Colomar: I think what matters are the type specifiers within the bit-field. If you did not spell it "int" the rule does not trigger.
Liber: And the behavior is?
Colomar: The same as everywhere else, it is not implementation-defined; it is just the type you got from elsewhere. So it is "signed".
Gustedt: Integer literals have the type signed int, I think.
Myers: It does not say signed in the table.
Dusíková: If you do anything different from C++, it makes shared headers really painful.
Bazley: The only way I see this resolved it to take a census of compilers to see what behaviors are and write a table saying what the behavior is in each case. But who is going to do this work? Without having done that work, the answers are somewhat meaningless.
Celeste: My company has done that work, I just do not know who in the company to ask to get that table. We have done that work because it is part of our compatibility setup for the compilers we support. I have asked internally.
Seacord: We are now moving on to Issue 1008 because we need more information to resolve this.
Straw Poll: Would WG14 like to leave the standard as is in issue 1007 from N3609? 2-0-2 + 8-11-0 = 10-11-2: no consensus
Celeste: We might want a constraint violation, but only for the case where the spelling is problematic.
McCall: Do we know what the existing implementations for typedefs actually do?
Krause: We just carry over if the signed keyword was in the typedef.
Straw Poll: Would WG14 like to have a constraint violation in issue 1007 from N3609? 1-1-2 + 8-10-3 = 9-11-5: no consensus
Straw Poll: Would WG14 like to change the behavior to reflect C++ in issue 1007 from N3609? 3-0-1 + 6-9-6 = 9-9-7: no consensus
Myers: Do we want to answer the specific questions for C23?
Bhakta: Whoever asked these questions is looking for clarity for C23, so we should answer them regardless of C2y.
Bazley: Should we ask Krause what SDCC does?
Bhakta: Krause is not the only implementer.
Seacord: In cases where there are no expectations for what the behavior is, we define the types to be the same as outside the bit-fields.
Bazley: To resolve this, someone needs to survey a bunch of compilers and create a table of current compiler behavior. Who does this benefit?
Celeste: I know my company has done this work, for compatibility with every compiler we know about. I have requested that info but would not get it soon.

- Issue 1008 (attributes with tags)

  Straw Poll: Would WG14 like to accept the suggested correction to Issue 1008 in N3609? Adopted by unanimous consent

- Issue 1009 (extern thread_local)

Celeste: I would be surprised if this changes any implementations.
Straw Poll: Would WG14 like to accept the suggested correction to Issue 1009 in N3609? Adopted by unanimous consent

- Issue 1010

  McCall: The wording needs to be specific about what is being described. What is the "last thread"?
  Bazley: Also, "is called" is misplaced in the wording.

- Issue 1011 (powr(-ve,qNaN))

  Straw Poll: Would WG14 like to accept the suggested correction to Issue 1011 in N3609? Adopted by unanimous consent

- Issue 1012

  Colomar: Since printf() only returns an error, you do not need to return a count. They should simplify the language to say "it fails".
  Bhakta: I prefer the proposed suggestion.

- Issue 1013

  Bachmann: For Question 1, I disagree, we cannot say that it is a bug.
  Ballman: We should have never specified these function type attributes. There is a reason C++ does not standardize type attributes.
  Krause: But "Attributes are ignore-able" does not capture reality.
  Bazley: If you apply the same concepts to type qualifiers, you get nonsense. If you have an attribute on one thing and not on another, you form a composite type. I was using composite type in GCC.
  Ballman: The standard talks about attributes in a vague way. We never say if we have scoped vs. non-scoped. Some decisions we make can apply to vendor attributes.
  Bhakta: For all the questions, they should all be the same type. Clang considers questions 1-3 as different types.
  McCall: With regard to vendor attributes, we have to have different rules from how we treat attributes formally in the standard. We should fix this in the long term, understanding that these things introduce actual complexity.
  Bazley: By definition, an attribute is ignore-able. The way they should be expressed should be very regular.
  Colomar: Things like the GNU path attribute are not really ignore-able.
  Ballman: Standard vs. vendor attributes are different beasts. We must be careful about how to deal with this. Clang has 400-500 nonstandard attributes. We should clarify the rules for standard attributes.

- **Gustedt**, Another daemon: waiting for condition variables [N3559] (0.5 hours)

  Svoboda: We could always state that if a mutex passed to cnd_wait() is previously unlocked, it remains unlocked until the condition is satisfied. Thus requiring that other threads not be blocked when the condition is satisfied should explicitly be undefined behavior. as suggested.
  Gustedt: No, it is still undefined behavior. Something destroyed by cond_destroy() means no one else can do anything on that variable.
  Bazley: Descriptions of mutex are always tricky. Should it say "shall have been locked"? Do you need to add "and not yet unlocked"?
  Gustedt: Yes, this is an ambiguity.
  Straw Poll: Would WG14 like to adopt N3559 as is into C2y? 2-0-1 + 15-0-5 = 17-0-6: consensus
  Seacord: Can we vote on adding this to obsolete versions of C?
  Straw Poll: Would WG14 like to adopt N3559 to obsolete versions of C? 2-0-3 + 17-1-3 = 19-1-6: consensus

Morning break

- **Múgica**, Phrase semantic boolean operators as if bool, v. 1.1 [N3602] (0.5 hours)

  Douglas: How much code would this paper break?
  Ballman: It should not break any code; the semantics should be the same. It changes semantics, and might break some Clang-tidy checkers.
  Bhakta: The section 6.10.2, paragraph 15 change needs to be reworded. For example, "convert their operands, or some of them, …" should be "convert some or all of their operands …".
  Bazley: If this has no semantic difference, the as-if rule should apply.
  Douglas: This paper proposes a bunch of stuff, but addresses no breakage. C++ does use bool for logical operators, why doesn't this paper compare its proposals to what C++ does? I would like to see more meat on the bones. I would feel much more comfortable if this was known to match C++.
  Celeste: If you are changing result types, that is observable, although changing operand types is not. Overall, this is still a good thing, as using ints instead of bools is now a thing of mockery.
  Colomar: The result type is in, covered by the as-if rule.
  Ballman: We would have to consider breaking code by changing the result type.
  Gustedt: The first paragraph is clear on semantics. We should first do a cleanup and then have a paper which changes semantics.
  Seacord: This talks about the conversion rules, which is unusual for operator descriptors. It is also Inconsistent with other operator descriptors in the standard.
  Straw Poll: Would WG14 like to adopt N3602 as is into C2y? 2-3-1 + 7-7-11 = 9-10-12: no consensus

- **Múgica**, Floating literals converted to bool [N3545] (0.5 hours)

  Douglas: It would be extremely wise to opt in explicitly. Implicit opt-in would be dangerous.
  Bazley: I am puzzled by the motivation. Presumably you could cast these expressions to int, not just bool, right? *no* I would expect that when casting a floating-point constant to int, you could then use it as shown in the paper.
  Celeste: 0.1 != 0 so conversion to bool is not obvious.
  Svoboda: The dialect makes code harder to read. Also newer languages like Java require explicit typing, going the other direction.
  Liber: Floating-point has different properties than integers. 0.0 and 1.0 do not represent true and false like 0 and 1 do. These are error-prone.
  Celeste: Since the restriction against floating-point-to-bool exists, we should keep it.
  Bazley: I loathe irregularity. I wish the motivation had been better explained.
  Straw Poll: Would WG14 like to adopt N3545 as is into C2y? 1-1-2 + 2-15-9 = 3-16-11: no consensus

- **Myers** Floating-point TS 18661 (C23 version, 2025) issue log, r1 [N3610]

  Straw Poll: Would WG14 like to accept the wording in reflector message 30750 for future editions of TS 18661-4? 1-0-2 + 13-0-9 = 14-0-11: strong consensus

- **Colomar**, Relax restrictions on standard attributes [N3631] (0.5 hours)

  Bazley: Different implementers are free to lay out structs in memory however they like. It all depends on the compiler and ABI and platform. So this path attribute is a hint to the compiler to do something differently.
  Bachmann: Split the standard for small platforms.
  Ballman: The assertion in the paper that these attributes can not be standardized, but that is not true. For example, see clang::requires_constant_initialization. WG21 liked this attribute and standardized it. My bigger concern is that this does not leave a good path for standardizing attributes in C and then C++. We intended attributes to be identical in C and C++, which this paper would discourage.
  Colomar: I dislike this for keywords.
  Celeste: If I understand correctly, the proposal is not to add attributes that can be ignored unless the

platform explicitly supports the attribute.
Bazley: Attributes that do not look like attributes are confusing.
Krause: I like the diagnostic requirement.
Bhakta: One thing missing is the "vendor" namespace attribute. That should be added.
Steenberg: I have seen reasonable speedup when serializing packed structures. That should not sway whether this is right for the standard.
Liber: Are there any issues with pointers in packed data structures with this document? *No, this is not about the "packed" attribute.*
Colomar: Existing implementations are allowed to ignore existing attributes and new attributes.
Ballman: A correct program with attribute remains correct if the attribute is removed. I am wary of going back on this without a lot of motivation.
Gustedt: If you remove "packed" in all your sources, the program is still correct.
***Ballman**: It depends on pointer arithmetic.
Bhakta: We cannot vote this directly in anyway.
Opinion Poll: Would WG14 like to accept something along the lines of N3631 into C2y? 1-1-1 + 5-8-10 = 6-9-11: no preference
Bachmann: I do not want to have the standard split for features and I prefer proper serialization.
Celeste: Something does not have to be standardized to be useful.
Bazley: I do not like diluting the notion that attributes are ignore-able.
Łukasiewicz: I would prefer "packed" as a keyword rather than an attribute at all

- **Gustedt**, Function call without decay [N3557] (0.5 hours)

  Krause: This paper has the same spirit as the other paper that was accepted. This one makes more sense.
  Bhakta: Does this mean you can pass functions to other functions, rather than function pointers?
  Gustedt: This is a rewrite of function prototypes rather than function calls.
  Ballman: I note the lack of implementation experience. Clang models all function calls as calls to function pointers.
  Gustedt: I disagree. You have a name for a function pointer, which is really a function. This is not support to change anything on the user's side.
  Ballman: OK, but this is work for no benefit. Clang's AST has function pointers, an implementation detail. Changing that would be expensive, and we probably do not need to change it even if this passes.
  Krause: You have decay in your AST and would not change it unless this goes into the standard. SDCC does not decay.
  Bazley: I worry that the language risks getting frozen because of the Clang AST.
  Gustedt: We never expect that Clang internals, including the AST never change between versions of Clang/LLVM.
  Ballman: Yes, you should assume that clang will change internals between versions. This invalidates a lot of function calls; it is more a matter of scale. Clang does not have any control of who uses the AST, but AST changes require maintenance of that code.
  Straw Poll: Would WG14 like to accept N3557 as is into C2y? 2-0-2 + 11-6-7 = 13-6-9: not strong consensus, not adopted.
  Ballman: What would make me change my vote from no to yes is stronger motivation in the paper.

- **Colomar**, Rename s/uimaxabs/umaxabs/ [N3577] (0.25 hours)

  Douglas: I think the new name "umaxabs" is prettier than the current name "uimaxabs".
  Ballman: Why do we keep adding things to the standard and then changing them later? This punishes early implementers. If anyone has released it, it is problematic.
  Bazley: It is already in glibc but people should be warned about changes like this.
  Ballman: They are warned, but this is an ABI-breaking change.
  Douglas: I could avoid breaking the ABI by aliasing the function.
  Colomar: This requires strong consensus. Glibc has not released this, it is just in the mater branch.
  Liber: I sympathize with **Ballman**, but that means we cannot put things in the standard until they are perfect.

Bazley: I do not think anyone voted against this based on the name. But requiring strong consensus will ossify the language.
Celeste: It is the difference between changing our minds and fixing a mistake.
Straw Poll: Would WG14 like to adopt N3577 as is into C2y? 3-0-0 + 21-1-5 = 24-1-5: strong consensus, adopted

- **Colomar**, Adopt qualifier conversion rules from C++ [N3629] (0.5 hours)

  Bazley: This is an interesting academic example, and relatively trivial. I do support it.
  Krause: Which qualifiers? Can I add atomic or not? *No, C++ lacks atomic qualifiers* It might also cause alignment issues. I would like a solution including "access qualifiers", perhaps for const, restrict, and volatile.
  Bazley: Calling a new term to replace "qualifiers" is a good idea. We should not block this. Currently C is a simple and regular language; it treats "restrict" like the other qualifiers. But this makes no sense for "restrict".
  Celeste: "Restrict" is a qualifier but not an access qualifier.
  ?: There are some wording issues to be resolved. It would be helpful for people if there were a few more examples.
  Opinion Poll: Would WG14 like to adopt something along the lines of N3629 into C2y? 3-0-1 + 18-0-5 = 23-0-6

Lunch Break

- **Bazley**, Type qualifiers do not apply to type categories [N3511] (0.25 hours)

  Straw Poll: Would WG14 like to accept the suggested correction in N3511? Adopted by unanimous consent

- **Colomar**, add streq() [N3611]

  Krause: New implementation details implies a cost, which may be small.
  Bazley: The strcmp() function's return value is not self-explanatory, unlike streq() and strne().
  Celeste: This is what I would like the CRFI process for. I would like to see a new section for utilities like this.
  Bhakta: If we move forward, I would like words that show how this compares to strcmp().
  Pygott: I can see why streq(NULL, NULL) should return True.
  Colomar: My paper is just a wrapper around strcmp().
  Douglas: How many github codebases use streq()? Apparently a lot of code does define it, with not necessarily the same semantics.
  Ballman: Not defining in terms of strcmp() actually makes this harder. A SourceGraph search yields 705,000 hits on streq(). While "str" is a reserved identifier, that has not prevented all these hits and code breaking if we adopt this.
  McCall: Many strcmp() implementations vectorize, which is likely to be more efficient.
  Bachmann: The compiler can decide whether to optimize via vectorization.
  ?: Might it be worthwhile to add an uglified version of this function name?
  Svoboda: Perhaps we need a subcommittee to settle on which string functions we should use?
  Ballman: I have seen at least 5 function signatures in the streq() hits.
  Colomar: Everyone uses the library in string.h. Some use wrappers around these functions. Wrapping around the standard string library does not generate conflicts.
  Gustedt: We should cease to add to Clause 7. We need a space for these proposed header files.
  Myers: It is more reasonable adding these to Clause 7.
  Bazley: My library does not wrap the standard string functions, except perhaps sprintf().
  Seacord: I count 4 string API's so far: string.h, Annex K, strv, and **Douglas**'s version. What is your API for?
  Bazley: String buffers.

Myers: I was for eliminating Annex K.
Ballman: I would like to see a study group.
Patricia: This feels small enough that we should not reject it because there are others in the queue.
Łukasiewicz: Before we add many new API's, we should look at the array types first.
Svoboda: **Bazley** has good motives for his designs for a string library. We need a subcommittee to help resolve these directions.
Bazley: I spent weeks studying these issues., there might be many directions to go.
Straw Poll: Would WG14 like to adopt N3611 as is into C2y? 2-1-1 + 8-15-6 = 10-16-7: no consensus
Opinion Poll: Would WG14 like to adopt something along the lines of N3611 but using underscore-uppercase versions into C2y? 8-14-9: no preference

- **Seacord**, generic_count_type, v1.0.0 [N3593] (0.5 hours)

  Gustedt: I dislike this. Achievement with type-generic argument means there is no type conversion. It is an error that shifting by negative numbers is undefined behavior.
  Krause: Just get rid of the undefined behavior. I do not see any motivation to make the argument a specific type.
  Bachmann: It does not change much by itself. If you make it unsigned, this does reduce undefined behavior. Using "-1u" is the conventional way to set on all bits in a word.
  Bazley: No one would ever want to rotate more than UINT16_MAX. I prefer size_t in this place.
  Douglas: They all take unsigned int at the end. Should they not all just match?
  Myers: The idea of having a fixed parameter to a type-generic function fa poor design. Introducing unsigned int might risk introducing rather than removing undefined behavior.
  Krause: We have similar cases for functions that are already in C23.
  Bhakta: The type-generic function rules would have to be modified if this was a generic function with a fixed argument.
  Ballman: Do we know if anyone has implemented or deployed this? I do not.
  Gustedt: **Meneide** has implemented it, but I do not know any details.
  Seacord: I think we should just put options on the table and get directions about which options WG14 likes.
  Opinion Poll: How would WG14 like to address N3593?
  1: No change: 1+3=4
  2: Same signature as in C2y, undefined behavior becomes modulo: 0+12=12
  3: Fixed unsigned type: 2+5=7

- **Meneide**, Thread Attributes, r1 [N3627] (0.5 hours)

  Gustedt: I like the idea of having these features. Much of it is not novel, due to pthreads. I am nervous about the size of the API. I would be more comfortable if we can split out basic things like detach, and have a "thread attributes" section.
  Meneide: I did ask for CRFI.
  Douglas: All of the items we agreed
  Banham; Are we in danger of creating a new threading API a la POSIX? How does this help us write portable code?
  Meneide: I built this on top of pthreads. It also builds on Windows.
  Ballman: Are these locale-sensitive? I hate locales. It is a big ask to have that much complexity. I would prefer to have only the native name functionalities.
  Meneide: The wchar_t and char are locale-sensitive. The native strings are locale-insensitive.
  Myers: I have two main design points: One is over-specifying things in giving enumeration values for the aligned types. The other concern is the interface.
  Bazley: It is a good design but over-engineered.
  Steenberg: You cannot actually use POSIX because POSIX's thread libraries are not compatible. Mixing them would be very dangerous.
  Bachmann: It is valuable to print the thread ID. But the name has no need to be internationalized.

Gustedt: It is difficult for us to agree on the thread-naming part. Having one simple naming function for strings would be good. Then come back with more detailed things.

Afternoon break

- **Meneide**, Transparent Aliases, r5 [N3487] (0.5 hours)

  Douglas: What happens with linkers that cannot do this?
  Meneide: Linkers do not have to do anything. This paper works entirely as a translation-time, runtime feature. It requires no extra memory or extra storage.
  Colomar: This does not really fix the linker issue. It does the same thing as #define already does. I do not see the reason for this.
  Meneide: Macros are nuclear weapons. Problems with macros immediately come to light when you use them this way.
  Bazley: In Graz, there was a poll that favored a typedef-style syntax. I hoped the next version of this paper would have presented that as an option. I still have misgivings about the syntax. D used to provide a typedef-like syntax, but now it looks more like this syntax. There was a third syntax involving "alias".
  Krause: This is a problem implementations need to deal with. We do not need to put this tool into the standard and expose it to users. I do not see sufficient motivation.
  Gustedt: This changes the grammar of the language in that we have special rules for typedefs, because the grammar is different if an identifier is a type or not.
  Bazley: This does set a precedent. This alias declaration looks like a declaration and initialization of an object. This is another way of hiding indirection or aliasing.
  Ballman: There are constraints that forbid one from transparently aliasing types.
  Meneide: It does not work for types because that is what typedef is for.
  Colomar: You can do that with _Generic today. Can you show an example of something that is not safe with a macro?
  Seacord: Everything is in the global namespace. You had an example: #define free(). Even inside a structure with a member called free, it would replace .free with `.__free`.
  Bazley: I want to believe macros are the solution, but I cannot.
  Uecker: I do not hate this paper, but it does not make sense to me.
  Celeste: The direction indicates pretty much for other languages with macros is that integration into namespaces is better than a separate preprocessing pass.

- **Meneide**, embed Synchronization, r1 [N3490] (0.5 hours)

  Ballman: I support adding the offset parameter. I also do not want to see C vs. C++ divergence in the preprocesor.
  Banham: Should we be using the term "byte" here? For files we could use "optics" or "characters"?
  Meneide: The wording says it behaves if it skips ahead a fixed amount of fgetc() calls.
  Vlad: Do you have plans to bring C++ recent changes in as well?
  Meneide: There are several CWE's, they changed the wording for how sequences are handled.
  Bhakta: Can you summarize if the changes here are minor?
  Meneide: Yes. They do not change how things work.
  Ballman: One thing WG21 did: they fixed the order of some of these parameters, right?
  Meneide: That did not get enough traction in WG21.

- **Myers**, Open Issue processing C23 issue log, r1 [N3609] (1.5 hours)
  - Issue 1016 (digit separators in `__LINE__`)

    Bhakta: Should we do this for other macros as well?
    Meneide: Have we considered a preprocessor link step that blows up all digit separators from numbers?
    Ballman: Do we even need to solve this problem?

Straw Poll: Are there any objections to adopt the suggested wording in issue 1016 from N3609 for C2y? *none*
Straw Poll: Are there any objections to adopt the suggested wording in issue 1016 from N3609 for obsolete versions of the C standard? *none*
Banham: A note to the editor:
Note to the editor: In N3550, issue 1016 uses the phrase "excluding any digit separators (6.4.5.2)". Please replace "excluding" with "not containing".
Seacord: Also note to the editor:
Note to the editor: Consider removing the parentheses.

- Issue 1018

Celeste: I do not believe that is what they actually are in C.
McCall: We should change compatibility rules to just consider the underlying type. People still use enums in ABI interfaces.
Myers: I am not sure this is guaranteed across different translation units.
Bhakta: Out of the alternatives here, the first one seems to be the most obvious, and the last one is the most useful ad flexible. The second one seems to be not useful or understandable and probably hard to implement as well. I would prefer 1 or 3.
Gustedt: I am not sure the proposed solution is one we want. This is too strong.
Ballman: It would be weird if enumerations behaved drastically different than anything else. I struggle to accept this, but I am sympathetic to it.
Bazley: The first solution seems tautological to me.
Krause: For a struct you do not know the members; you do not know how much space to allocate. But you know how much to allocate for enums.
Bazley: Do we really pool enumerations as numbers? They do not behave like numbers.

Wednesday, 27 August

- **Seacord**, Integer Sets, v2 [N3644] (0.5 hours)

Svoboda: Is the point to adopt C++ terminology for bool?
Seacord: The point is to move closer to C++'s way of doing things.
Ballman: We had made BitInt signed, but BitInt did not have a sign of 1 because there are no value bits; it only supports 0 and -1. However, I am in full support of the current terminology which adds BitInt<1>.
Myers: It is not a good idea to rewrite paragraphs about bit-fields. Some of the confusion is because we have signed and unsigned referring to types that do not behave like signed and unsigned. They are subsets of the types whose behavior is signed or unsigned.
Bazley: The taxonomy should clarify the standard. I am in favor of this change.
Ballman: We could bring the paper to SG22, the C/C++ compatibility study group.
Bhakta: In the section 6.2.6.2, paragraph 1 change, why have "least significant bit"?
Seacord: I agree; it is been removed in a newer version.
Bhakta: Why the 6.2.5p20 addition?
Seacord: I defer to the committee for consensus on this paragraph.
Kaye: This is a sensible change.
McCall: I like this paragraph, I think we need it.
Uecker: It is important to say something about bool.
Gustedt: This is in paragraph 1.
Seacord: It feels like consensus is to eliminate this paragraph.
Svoboda: I sure would like to see those graph partitions in the standard text.

- **Uecker**, Earthly Demon XV: Definition of Main (Updates N3562) [N3623] (0.25 hours)

Krause: It is common that a user supplies a little code that passes control to a startup function. The compiler does not know anything about the startup function. I do not want this for freestanding

implementations.
Ballman: Overall, I am in favor of the intent. But it is moving farther from existing practice. "main()" is not the way that everyone defines their startup function. I would like to see us admitting real-world practice. The new constraint is correct, but also weird when it is an implementation-defined type.
Bhakta: I care about obscure platforms. This is too specified, there are many ways to enter a program.
Bachmann: There should be no restrictions for freestanding platforms. We often have a "pre-main" to switch clocks.
Krause: Often our platform will write an assembly function and call it from C. It would be ridiculous to have every assembly function with a command-line parameter. It provides no additional safety, because it is only added to this one function.
Ballman: Currently, the type is implementation-defined. The name should be implementation-defined as well.
Celeste: We wasted many hours on this in MISRA. Striking this entire section from the C standard would be appropriate.
Bazley: Failure to standardize makes this committee pointless. Any compiler that does not let you run a "hello, world!" program with main() should be nonconforming.
Ballman: **Celeste**, that is an interesting way to think about this.
Uecker: The other ways of naming main() are out of scope of my paper. It constrains the type.
Bachmann: This does not describe embedded systems appropriately.
Gustedt: I propose a compromise: It wraps up things for hosted implementations, and makes a new claim for freestanding implementations. The freestanding is not complete, and needs a new paper. We can add the constraint for hosted implementation. Add "for hosted".
Uecker: I am fine with this wording change: "Change paragraph 1 to "For hosted implementations, the function called at program startup shall have one of the function types specified in 5.2.2. There shall be a definition of this function with external linkage."
Straw Poll: Would WG14 like to adopt N3623 with the wording change into C2y? 2-0-1 + 15-3-6 = 17-3-7: strong consensus

- **Uecker**, Representation of Pointers and nullptr_t [N3563] (0.25 hours)

  Gustedt: I am in favor of this. Now is the time to clean this up.
  Celeste: This is a great usability change.
  Straw Poll: Would WG14 like to accept N3563 into C2y? Adopted by unanimous consent

- **Ballman**, Member access of an incomplete object [N3532] (0.25 hours)

  Douglas: It gave me funny vibes that there may be a use case we have not thought of.
  Ballman: Struct members used to be globally available. So this was a holdover.
  Celeste: I asked someone on my team; I do not require this to be supported by the standard.
  Straw Poll: Would WG14 like to accept N3532 into C2y? Adopted by unanimous consent
  Straw Poll: Any objection to adopting N3532 into previous versions of the standard? Adopted by unanimous consent

- **Coates**, Slaying some earthy demons - remove UB 28, 29 [N3565] (0.25 hours)

  Svoboda: If this is a "ghost" undefined behavior as **Myers** suggests, how does the paper change?
  Myers: Some sentences go away.
  Svoboda: But that could be a future paper. We can still vote on this paper.
  Gustedt: We still have numeric indexes that are not allowed in identifiers, but some implementations, such as Clang, allow them, and we allow an extension paper. I prefer these to be implementation-defined rather than constraints.
  Krause: It is common for implementations to allow extra characters, such as $.
  McCall: The sentence about $ could go into the Constraints section.
  Ballman: Have you verified this does not make some odd incompatibility with C++. I can connect you with someone in SG16 who can answer.
  Banham: I am confused by **Gustedt** wanting Unicode characters outside this set. But **Coates** is moving

towards constraint violation; noncompliant compilers will not abide by it anyway. The Unicode standard is evolving.
Gustedt: And the C standard is not pinned to any particular version of Unicode.
Bhakta: It is not the compiler that chooses. The standard indicates 10646 (Unicode).
Ballman: We cannot use the Unicode definition of what are valid chars in an identifier. We have made some explicit exceptions, including $.
Celeste: I am not convinced we need undefined behavior for the extension point. It is necessary to make this undefined behavior for the extension.
McCall: Implementations need to add to the set of identifiers. Any implementations want to maintain different sets of Unicode tables, which would be required if each standard encodes a particular version of Unicode. That would be a huge amount of work for compiler vendors.
Seacord: We have changed Annex D, eliminating section 2.2 during defect reports. Our change was a band-aid before a complete rewrite. It was trying to follow C++. Things are in transition, and it is unlikely for this to survive without changes to C2y.
?: The keyword is "profile". Make it implementation-defined. There is a medical profile for people who like subscripts.
Opinion Poll: Would WG14 like to adopt something along the lines of N3565 but using implementation-defined behavior rather than constraint violation into C2y? 1-0-3 + 13-1-9 = 14-1-12: strong direction

Morning break

- **Coates**, Slaying Some Earthly Demons - remove UB 30 (0.25 hours)
  - approach 1 [N3603]
  - approach 2 [N3604]

Celeste: I prefer approach 2 (that eliminates "significant characters")
McCall: We do not meaningfully define what happens when you exceed the limit of significant characters.
Krause: I disagree that every program that would be broken is already wrong. But I do not control my linkers; we use third-party linkers. Either two identifiers will be different or the same. I could work with "implementation-defined".
Bhakta: I prefer approach 2. It does remove the implementation documentation burden.
Ballman: I prefer approach 2. It would be the linker that would document this stuff.
Bachmann: Is there code with such large identifiers?
Gusted: Yes, if you use UTF8, perhaps also auto-generated code.
Bazley: I prefer if programs that could be linked wrongly be diagnosed. I would expect the constraint to be the object file format. It seems strange that a linker can ignore some "non-significant" identifier characters.
Douglas: Do we not feel that 63 significant characters is too short?
Uecker: The linker is part of the implementation. We need some better solution, perhaps working with linker developers.
Colomar: I would change from undefined behavior to a constraint violation.
Bachmann: We have heard that these limits cannot be changed. But they have been changed in the past.
Celeste: Is 63 characters exceeded? Yes, easily by auto-generated code. Right now, this does not need to be diagnosed.
Ballman: I am sympathetic to **Uecker**'s point. Unlike compilers, linkers can be dumb, so they require little maintenance. I do not know how to resolve this. Undefined behavior might be the most reasonable choice if we cannot guarantee how the linker will behave. Are there CVE's with this?
Uecker: What does Rust do?
Łukasiewicz: If the name exceeds the limit, the significant bit of length exceeds the limit. I favor option 2.
Bazley: I can talk to some linker people and see what they think. We may need to establish a linker standard.
Krause: Do users really use long identifiers? I know two projects who used long interfaces to make them restrictive but made sure to keep the beginnings distinct.
McCall: The common linkers really do not have identifier limits.
Bhakta: There are many limited linkers out there. Some pick the first identifier without further checks. So

you cannot force compilers or linkers to diagnose this.
Bazley: The thread is being lost. If linkers have limits beyond the standard, they should produce diagnostics.
Tarditi: You cannot tell people their conforming implementation is no longer conforming. You can have 8,000 identifier lengths. We do not control linkers, and no one wants to retrofit their linker.
Wiedijk: There is a file system that still has 11-char limits. C is still good for quick and dirty things. Raising the requirements on ISO C feels like ignoring the little guy, especially compared to C++. I prefer option 1.
McCall: There are implementation techniques that does not run into significant limits. You hash the symbol name, which has problems but does not break code.
Liber: I do not like arbitrary limits. We should not be dictating what those limits are.
Opinion Poll: Regarding N3604, which approach does WG14 prefer?
Do nothing: 1+6=7
Implementation-defined: 0+4=4
Remove significant characters: 4+13=17

- **Gustedt**, static_assert without UB [N3525] (0.5 hours)

  Ballman: An implementation is allowed to extend what an integer constant expression actually is, right?
  Gustedt: You are not internally consistent if you do that.
  Douglas: How does this compare with C++'s static_assert? Would C code with static_assert() blow up in a C++ compiler?
  Gustedt: I do not know.
  Ballman: C++ gets contextually converted to bool, but you can have a constant expression that is not an integer constant expression. As we continue to expand constexpr, if I have a pointer that can be NULL that is not an integer constant expression.
  Straw Poll: Would WG14 like to adopt N3525 as is into C2y? 2-0-2 + 16-3-4 = 18-3-6: adopted

- **Mailhol**, Static assertions in expressions, v2 (updates N3538) [N3637] (0.5 hours)

  Krause: This is a niche use case. For lambdas we would not need this. You could use a normal statement. I do not want this in. If something like lambda goes in, perhaps we can reconsider this proposal.
  Bazley: I like this proposal.
  Celeste: I do not think should depend on a more complicated feature. It is almost like an arbitrary missing feature to static_assert().
  Colomar: I have a use case. I wrap string functions in macros. I need static assertions to assert that an expression is an array (not a ptr). So I need this.
  Ballman: Is C++ taking a similar feature? *No*
  Uecker: Would the block actually break something? *Yes*
  Łukasiewicz: This brings static_assert() closer to normal assert(), so I like it.
  Omar: This version would not be usable in an integer const expression because it has void type, right?
  Mailhol: I wanted to keep it simple. If accepted, my next paper would be to propose this.
  Opinion Poll: Would WG14 like to adopt something along the lines of N3637 into C2y? 2-0-1 + 18-4-5 = 20-4-6: strong direction

- **Douglas**, Standard secure networking [N3533] (0.5 hours)

  Kaye: Would this be mandatory or optional?
  Douglas: Optional. Some C implementations would have no network.
  Myers: This should go to CRFI, it is more appropriate there. It also needs normative references, and a lot of other new concepts.
  Krause: Networking is important but not trivial. The WG14 committee needs experience with networking, or else offload this proposal to a subcommittee.
  Wiedijk: It should be a TS first.
  Bhakta: C++ did try the TS route. Did it get adoption in the C++ community?
  Douglas: This is the post-TS implementation, after the TS was killed.

Liber: Are there dependencies in your library?
Douglas: I think not. All I/O go through struct io_vec.
McCall: Is there any platform API author feedback?
Douglas: That was not necessary; this is as ABI-tight as you can imagine.
Krause: It might bring in many identifiers, many prefixed with "tls". Internally, I would like a prefix to avoid namespace pollution.
Ballman: I am concerned in that secure networking API's have changed a lot in the last 20 years. What chances are there of us standardizing this, and then discover that implementations have moved on?
Douglas: We are only supporting TLS, no other protocols. I think it will be around forever. I claim that they have fully debugged SSL by the time TLS 1.3 came out.
Krause: Is TLS ready for quantum crypto?
Douglas: In TLS you list which crypto algorithms you support in the protocol. If both endpoints agree on a crypto algorithm, they use it So TLS is future-proof with regard to crypto.
Colomar: I do not think we should have secure networking before we have regular networking.
Dusíková: You want secure networking by default.
Bazley: I am tempted to say it is too low-level. One part of ISO C I most value is strings. If this were more well-integrated with files it would be compelling to me.

Lunch Break

- **Douglas**, Standard secure networking [N3533] (0.5 hours) (cont)

  Seacord: This is a strong candidate for CRFI, but it is not right for anything else. That is the obvious route for this.
  Celeste: CRFI has a proposed extended library section, for things not quite suitable for a TS. So this is very permissive for CRFI.
  Bazley: Why use function pointers? It is not like any interface in our standard library.

- **Meneide**, Integer Constant Expression-Initialized const Integer Declarations are Implicitly constexpr, r1 [N3600] (0.5 hours)

  Ballman: Overall I am in favor of this.
  Celeste: I am wary of block D5, it is not obvious whether it is going to be one or not. Where it is not explicitly static, it depends on what it is initialized with.
  Uecker: I also support this paper; it constraints things that should be constrained. I am concerned about the case of propagating const from the initializer.
  Bhakta: This broke some users' code. Have you heard the other way around?
  Ballman: My mental model is that literal and constexpr when properly initialized are interchangeable.
  Bazley: C++ is discussing if they should deprecate this. Why?
  Dusíková: There is a draft in the C++ library group, but it is only for analysts and integers.
  Gustedt: I think C++ has different semantics here. At file scope, it is automatically static.
  Opinion Poll: Would WG14 like to adopt something along the lines of N3600 into C2y? 1-1-1 + 22-1-7 = 23-2-8: clear direction

- **Meneide**, Additional Half-Open case Range Syntax, r1 [N3601] (0.5 hours)

  Bazley: This is an excellent proposal. But I am ambivalent about it; it feels redundant. You do not get errors about incorrect bounds statements in C, why get them here?
  Krause: Imagine that you have an if condition that cannot be taken because you swapped your bounds. We already have if and switch, now you want half-open on the right side. Why not half-open on the left side? We do not need every detail to formulate a switch statement or if/else chain.
  Liber: It is all throughout the C++ library. I might want ranges from something high to something low. But that is a violation here.
  Colomar: It is more of a footgun than what we have now. A user would expect the same behavior from a

half-open range vs. a closed range.
Bhakta: Is there any C experience with this?
Meneide: No
McCall: In Swift, half-open ranges are more important than closed.
Coates: The history seems to be that closed ranges were used extensively and automatically generated. Is this intended to sidestep that issue to have tighter diagnostics now?
Meneide: The proposal that added closed ranges to C did tend to use this. The reason we decided not to use constraint violations was configurability / code generators.
Myers: This is a somewhat marginal feature.
Ballman: I agree with the importance of half-open ranges, but it does not need new syntax. Ranges inside cases are a small subset of what people are using. I do not think we need it yet.
Colomar: I am not opposed to the idea that they are more common than closed ranges. But I would like to see a proof first. We should have the same constraint violations as closed range.
Wiedijk: I agree that the constraints should be same for open vs. closed ranges.
Simerda: In the editorial, why not include the range type from Python library?
Bazley: Later we will want initializer ranges, which will be influenced by this paper.
Opinion Poll: Would WG14 like to adopt something along the lines of N3601 into C2y? 0-2-1 + 12-12-6 = 12-14-7: no direction

- **Meneide**, Literal Suffixes for size_t, r2 [N3485] (0.5 hours)

  Myers: I have a problem with using "may not" in a note due to its grammatical ambiguity.
  Bachmann: Size_t is a typedef name for another type, and we do not have literals for other non-types. This is inconsistent.
  Krause: Only 10% of the C++ paper applies to C.
  Celeste: I am surprised by its absence in C.
  Colomar: I am also surprised. I think I do not see any use for it.
  Bazley: "Ease of library independence" is in the C charter. I do not care deeply about it. It seemed weird to have a suffix for literals that have a type only available as a typedef.
  Bhakta: Is there implementation experience?
  Meneide: Except for Clang and some that have an extension, no.
  Ballman: In Clang, we do not expose it as an extension, but as a C++23 feature. Types are either important or not, and users like literals for their types. There are users who want their types to match because conversions are complicated.
  Liber: The language returns size_t so people are surprised when the feature is not already there.
  Gustedt: size_t is a typedef of a type that exists. I would be more in favor if this were consistent with printf specifiers. Having more cases is just more to learn.
  Celeste: Users do not want to do conversions if they can avoid it. Rust does not allow conversions at all.
  .Colomar: I see why you want this for size_t but I also see why you might want this for time_t. Why size_t and not time_t?
  Ballman: Size_t is used much more than the other extended type. So it is more special.
  Seacord: "The corresponding signed integer type for size_t": that type does not exist in the standard. I would prefer that that line said "a nonstandard integer type of size_t".
  Colomar: You can define a signed version of size_t using _Generic.
  Bachmann: One of our jobs is to say "no".
  Douglas: I am strongly in favor. I cannot write a size_t literal without a cast. Not having a size_t literal in braced initializer list, you need size_t casts.
  Straw Poll: Would WG14 like to adopt N3485 with the editorial correction into C2y? 1-2-0 + 18-5-5 = 19-7-5: weak consensus, not adopted
  Seacord: Reflector message 25775 has **Bhakta**'s task and script for determining consensus

Afternoon break

- **Douglas**, Lingua franca Results [N3599] (0.5 hours)

Celeste: I strongly support this. It is great to start with C90 first.
Bhakta: Why did C++ reject this?
Douglas: When Herb Sutter's paper was not going to happen, this got de-prioritized.
Krause: I do not want this in the language. I want it in the standard library. Go all the way and use the "std" prefix on everything in the paper.
Bachmann: The paper is a bit sparse. I am curious how this works in Fortran and other languages?
Svoboda: What about the C standard library. Are there return values?
Douglas: This is not planned for the C standard library, but it is easy to do for interoperability with other languages.
Svoboda: Java and Python have Option and Result types and no one uses them.
Bachmann: On POSIX I can augment errno with, say, a file descriptor. Can I include that in the error?
Bazley: Any largish type-safe program needs an error type. For C should be a struct. Large projects are troubled by having a single list of errors. Also, where does the unique ID come from?
Douglas: We want to not destroy any error info, but encapsulate it.
Colomar: This looks like an XKCD standard problem. I do not see how this makes my life easier.
Łukasiewicz: I like the idea of the Result type. Making it a library will make it cumbersome to use. This requires more typing and attention.
Ballman: It seems like the lingua franca are centered around modern languages. Is there any experience with Fortran or Cobol or older languages?
Douglas: If it can work with C, it is good. No one has reported issues. There was one person who was using it with Fortran.
McCall: The representation of error values is always type-erased.
Douglas: It can go into erased form if it will fit in memory. The C++ paper has full wording for C++. There is also a reference implementation.
Bazley: Yes, std::result_destroy. I prefer to avoid memory management when possible. Is there any cleanup? I do not want an English message, I prefer a token that works in any language.
Opinion Poll: Would WG14 like something along the lines of N3599? 5-1-0 + 15-3-4 = 20-4-4: strong direction

- **Colomar**, Restore the traditional realloc(3) specification [N3621] (0.5 hours)

  Seacord: We could not tighten up the language because of OS implementation divergence, so we loosened the standard, and eventually gave up and made it undefined behavior. I am thinking of proposing a new realloc() function with the original intended behavior: realloc() returns NULL for error and for size=0, it return a 0-byte block. If the block fails, it returns NULL. We cannot change the existing behavior of realloc() because that breaks lots of user code. I prefer deprecating realloc() in favor of this "new_realloc()" function.
  Colomar: I was careful not to break any existing code. BSD's realloc() is not broken, so they will not change.
  Ballman: It is incorrect to say this change does not break code. It does not change the semantics of code, but small memory leaks constitute breakage.
  Steenberg: A lot of people depend on using realloc() as free() so they expect it to return NULL. Changing this will break their code.. And a new_realloc() function is a big ask.
  Colomar: Code that hard-codes with free() but returns NULL is broken today.
  Bazley: I strongly support this paper. I do not think we can replace realloc() any more than we can replace atoi().
  Celeste: It is community-knowledge that we need to un-teach that realloc(0)==free(). Making it undefined behavior made some people angry. My constituents see undefined behavior and think "error". We could use this anger as leverage.
  Colomar: Memory leaks only happen if realloc() can fail for a size of 0.
  Bhakta: I am strongly against this. I like the idea of a new function.
  Colomar: If you take BSD code and run it with glibc, that is remote code execution.
  Svoboda: I detest code that uses realloc() in place of malloc() or free(); it makes the code impossible to static-analyze for memory leaks, or audit.
  Douglas: POSIX for realloc() says "Ignore what we say, do what the C standard says". I now think we

should put this back the way it used to be, and mark it deprecated ASAP. It feels solvable, but I suspect it is not.
Steenberg: If this was in the standard, many implementations would not implement it. That is why we took the decision last time.
McCall: I do not find the argument that compilers can reliably catch this convincing. If we want to make this change, the less problematic approach is to return something that does not need to be freed.
Celeste: I suggest we used a giant hammer to undefined behavior, we can do another one in obsolescent functions.
Eric: If we do a replacement interface for realloc(), you could choose at compilation time which realloc() you want. Portable code avoids realloc(0); they replace realloc(0) with realloc(1).
Ballman: The one thing that concerns me about having realloc(0) that does not require freeing is in getting a unique pointer value.
Krause: We will also follow the standard if this goes in, because we have very few users that use realloc() anyway.
Bazley: People who are against this paper should reply to the research that **Colomar** has done. He has engaged with the specific use cases.
Svoboda: Do not use realloc(0) or realloc(NULL).
Straw Poll: Would WG14 like to adopt N3621 into C2y? 3-2-0 + 6-11-6 = 9-13-6: no consensus
Opinion Poll: Would WG14 like to adopt something along the lines of N3621 into C2y, replacing realloc() with a new function name? 4-1-0 + 17-0-5 = 21-1-5: strong support

Thursday, 28 August

- **Gustedt**, Add type-safe minimum and maximum type-generic macros [N3543] (0.5 hours)

  Krause: I like it. But I prefer the name stdc_min()
  Colomar: Would this surprise anyone? Maybe we should recommend that implementations should diagnose if arguments are not literals and have different signs.
  Gustedt: We often do min with integer literals.
  Svoboda: Why is min/max insufficient?
  Gustedt: ISO C has min/max for floating-point, not for integers.
  Bazley: What is worse than the standard C promotion rules is inventing new promotion rules. I like the design. The "ckd_" prefix could be misleading. I prefer "std_" or "stdc_".
  Ballman: Let us not use the "ckd_" prefix. For integer types, there is no error. For floating-point types you could lack a type to represent all values if the arguments are different types. But we do not cover floating-point types yet. There may be error cases for implementation-defined integer types.
  Gustedt: I get big compilation types trying this on BitInt types.
  Myers: NaN's would be handled differently than IEEE 754. Also positive vs. negative 0. It is a bad idea for the proposal to address floating-point differently than IEEE 754.
  Gustedt: This is more a language feature than a macro. For floating-point integration I agree, we should remove it for now. For constexpr we can do constexpr in library macros.
  Tarditi: I support this, it is easy for people to make mistakes. If possible, let us keep floating-point out. Floating-point-to-integer conversion is tricky.
  McCall: You do not specify what types this works over. Since you store the result in one parameter, that means you must be explicit about the result type. If this in the ckd_ space, it should fail if the answer cannot be stored in the result type, like the other "ckd_" functions.
  Gustedt: For now we should constrain to integer types.
  Colomar: If this were in signed integers with infinite ranges, it would still work.
  Bachmann: Does this also support BitInts?
  Gustedt: Yes. There is a hard limit of 512 bits because Clang cannot see that the result will be small. That is an implementation problem.
  Kaye: Not doing error checking is more ergonomic.
  Seacord: I would prefer to see this not defined for bool types. I prefer bool not having mathematical relationships, outside of logic. Before C23, this would fit in with our proposed "cmp" prefix.

Uecker: **Myers** has an example with float vs. int64 error in the chat.
Bachmann: We cannot change integer promotion rules; they are deeply baked in. Changing them would be a complete mess.
Tarditi: I did not understand typeof<a+b> in the paper.
Gustedt: That was informal, not official C code.
Colomar: If you return the value and it is not returned by a pointer, what type will store it?
Gustedt: I expect the user will not need to store an error. In other places you could use auto.
Ballman: If a+b fails, then would not min(a,b) also fail?
Wiedijk: It should just be called min and max.
Gustedt: No, max is too often a variable name or macro.
Opinion Poll: Would WG14 like to adopt something along the lines of N3543 for integer types into C2y? 3-0-1 + 19-0-5 = 22-0-6: clear direction

- **Gustedt**, Clean up atomics, non-normative changes v4 [N3606] (0.5 hours)

  Myers: This is very close to being ready but there are a few places where wording is not quite right. There was an issue with prefix_increment wording. And one of the comments in an example was incorrect.
  Ballman: Have you talked with anyone in SG21 in WG21 to make sure we are in agreement?
  Gustedt: The specifications are on different levels.
  Opinion Poll: Would WG14 like to adopt something along the lines of N3606 into C2y? 4-0-1 + 14-0-4 = 18-0-5: clear direction

- **Douglas**, Modern signals handling [N3540] (0.5 hours)

  Colomar: What did POSIX think?
  Douglas: They loved it.
  Bhakta: Why did C++ not want this?
  Douglas: There was major opposition from one senior member.
  Ballman: Is the big problem to make signals and threads play well together?
  Douglas: Yes, there were problems with threading. It is now guaranteed thread-safe. It also lets you do thread-local signal handling.
  Svoboda: Windows signals can be re-sent, overriding your handler. This makes signals unworkable on Windows.
  Douglas: This is disabled on Windows; it does nothing. They are badly emulating POSIX, signals work poorly, which they have done since win32. That is why Microsoft loves the proposal.
  Bhakta: Has this been considered for a TS or CRFI?
  Colomar: Part of this comes from POSIX, and part is an extension. Could we split the POSIX paper from the extension paper?
  Douglas: Yes, we are adding sigset() from POSIX,
  Krause: Thanks for reusing thrd_ prefixes. Can you add a brief reserve prefix to the namespace?
  Gustedt: What platforms does the reference implementation run on?
  Douglas: POSIX, Windows, MacOS
  McCall: Minor, but in your thread_race, you have several void pointers with OS-specific definitions. Could those be more specific?
  Bazley: Could you not hide function pointer types behind typedef? It obfuscates the fact that it is a pointer. If a library maintainer ever wants to make this self-documenting, they might want to qualify that type as optional.
  Douglas: Do others agree? *divided reaction*
  Seacord: I asked Bill Plauger his greatest regret a long time ago, and he said "Adding signals to the C standard".
  Opinion Poll: Would WG14 like to adopt something along the lines of N3540 into C2y? 6-0-0 + 13-2-4 = 19-2-4: clear direction

Morning break

- **Múgica**, Classification of the register storage-class specifier [N3544] (0.25 hours)

  Kaye: "auto" is still in the storage durations.
  Seacord" That is for C++. We kept and extended the meaning for auto.
  Straw Poll: Would WG14 like to accept N3544 into C2y? Adopted by unanimous consent

- **Steenberg**, Effective Type in C [N3519] (0.5 hours)

  Seacord: The paper is OK. I have previously communicated some things that are incorrect. If there is an ambiguity, I would love to see a paper that resolves it, and an ambiguity would not surprise me. It is not obvious what the purpose of this paper?
  Steenberg: The first goal was to understand effective types. I shared the paper by request. There are three possible resolutions: 1. if you write using a member of a struct, you set the struct. 2. If you write to memory with a member of the struct, all the members get the effective type of the respective members. Does anyone argue that the type is the structure type?
  Wiedijk: I am trying to revive the Memory Object Model SG. I would love to discuss this in the SG. There is a mailing list.
  Uecker: I agree. The old SG started to discuss these topics. There are existing defect reports, but they were never fixed. But we have not even fixed provenance.
  Ballman: I am still struggling to understand why this is unclear. You get the effective type of whatever you are assigning from.
  Gustedt: The problem is that people tend to declare variables of struct type, then initialize members individually. At which point does this become the struct type?
  McCall: I do not know if it is supported, but it would be useful to say that an access to a struct member is also an access to a struct object for aliasing purposes.
  OPpinion Poll: Are there any objections to clarifying N3519? Adopted by unanimous consent

- **Celeste**, Simplified lexical scope for labels, v2 [N3658] (0.5 hours)

  Myers: I am not convinced the complexity is worth it.
  Celeste: It was intended to reduce complexity.
  Gustedt: I agree, this is simpler. If you use such a label in a goto, the label must be uniquely in that same function.
  Ballman: The Clang implementer also thought this was simpler.
  Bhakta: This is great for understanding. It may cause implementation issues. The use case matters more than the implementation.
  Bazley: Implementations aside, this complicates anything that complicates C code. It makes a simple Python-esque dictionary of goto labels insufficient.
  McCall: Is there a constraint about multiple loops in a hierarchy?
  Celeste: This does not enable that.
  Colomar: The only use for this is named loops. It is inconsistent that I can repeat names. I want to kill named loops. Functions are the names of loops.
  Opinion Poll: Would WG14 like to adopt something along the lines of N3658 into C2y? 1-1-0 + 17-6-1 = 18-7-1: strong direction

- **Thomas**, frexp and double-double (updates N3357) [N3535] (0.25 hours)

  Banham: Is floating-point not defined by the IEC standard?
  Bhakta: Not necessarily. You do not have to follow that to comply with ISO C.
  Straw Poll: Would WG14 like to accept N3535? Adopted by unanimous consent

- **Lenga**, Adding struct enum: Strongly-Typed Enumerations for C [N3568] (0.5 hours)

  Krause: Namespace pollution is the problem, but I do not think this solves the problem. Constants are the only motivation for me. And C++ compatibility.

Celeste: I do not like that you are using a different member access operator. The constexpr struct workaround is adequate.
Ash: I like the idea. If it is intended to be compatible with C++ it should go the whole way.
Kaye: I second **Celeste**. I see no mention of specifying underlying type, which we can do now in normal enums.
Vlad: Without specifying an underlying type, this looks like C++.
Łukasiewicz: What was the design for making it enum struct or class?
Gustedt: This paper does three things: It puts named scope around constraints, enhances the . operator to do something where the first thing is a type, and finally, it provides a new memory model for a new kind of enumeration type. I am opposed to that third thing. For the second thing, we should use a general new operator, like "::". I would be in favor of the second thing.
Coates: I like the idea, but not in its current form. I wonder if enum structs would get the type specifier **Kaye** suggests. I also wonder if you would end up with codebases with two forms of enums. If I want scoping control, and want to specify underlying type, I would have to do it manually.
Colomar: In C++, the enum class is a different type that cannot be mixed with integers. It does not look like C enum types. It would be surprising for C++ programmers, so it is a footgun.
Bazley: I should be in favor of this, but if I want enums to be strongly typed, I would wrap an enum in a struct. In C++ struct types are not separate namespace, what about struct-enum types?
Ballman: C++ enum structs evolved from putting an enum into a struct. We do have a little precedent in C for using :: with attributes. There is no name mangling involved; it is just a lookup operation.
Tarditi: I am in favor of this proposal; it cuts down errors. We should make sure it is compatible with C++. I know the wrapping-in-a-struct trick, but that breaks ABI's. Structs have different calling conventions than ints.
Bhakta: I disagree with the :: operator. I think users are more used to the . operator.
Svoboda: Does this make obsolete current enums?
Opinion Poll: Would WG14 like something along the lines of strongly-scoped enumerations similar to what is in N3568? 0-3-0 + 14-6-7= 14-9-7: weak direction
Opinion Poll: Would WG14 like to see something along the lines of N3568 into C2y? 0-3-1 + 9-5-11= 9-8-12: neutral divided response

- **Lenga**, Additional String comparison functions to complement strcmp() [N3455]

  Krause: We have seen an streq() function before. You added a performance argument. strcmp() could be optimized by compilers to match your streq() performance.
  Ballman: I did a global search for these interfaces. "str" prefix has 9,000 uses.
  Celeste: We can later handle questions about adding this to a CRFI library.
  Seacord: Earlier we had a streq() paper (N3611). Please work with **Colomar** about producing a common paper.
  Bazley: The motivation for adding functions to detect prefixes and suffixes is stronger than the equality functions. I propose shorter names for prefixes and suffixes: "pfx" and "sfx". The names should be mnemonic, rather than long. Also, if you add functions to work with char strings, you should add analogous wchar_t versions.
  Colomar: About "wcs" functions, I have the paper and can add those as a separate paper. We have OS matches with less conflicts.
  Seacord: If there is a paper for a narrow string version but not a wide string version, I would not vote for it in the hopes that the wide string version comes in later.
  Ballman: I did a quick search for **Bazley**'s suggestions: I have 800 hits for strprefix and 100 for strsuffix.
  Kaye: I disagree, if something stands out be being able to be read, that is good.
  Opinion Poll: Would WG14 like to see something along the lines of N3455 into C2y? 2-2-0 + 8-4-12 = 10-6-12: weak direction to proceed
  Opinion Poll: Would WG14 prefer to see "sfx" instead of "suffix" in N3455 and the same for "pfx" instead of "prefix"? 1-0-1 + 8-7-7= 9-7-8: split, no direction

Lunch Break

- **Thomas**, Clarify wording for 7.3.9.5 - cproj [N3536] (0.25 hours)

  Krause: We do not capture the topological properties of the Riemann sphere, but we also do not do this in floating point. I still like the change, I would add a footnote to explain how to model this on the Riemann sphere.
  Wiedijk: The Riemann sphere is one infinite point.
  Straw Poll: Any objection to adopting N3536 into the standard? Adopted by unanimous consent

- **Thomas**, Correct and clarify 7.3.1 Introduction of Complex arithmetic [N3537] (0.25 hours)

  Straw Poll: Any objection to adopting N3537 into the standard? Adopted by unanimous consent

- **Thomas**, Clarification for complex suffix specification [N3500] (0.25 hours)

  Straw Poll: Any objection to adopting N3500 into the standard? Adopted by unanimous consent

- **Thomas**, Make text consistent between creal & cimag [N3598] (0.25 hours)

  Straw Poll: Any objection to adopting N3598 into the standard? Adopted by unanimous consent

- **Gustedt**, Retire the concept of consume operations [N3607] (0.5 hours)

  Wiedijk: The rule "a synchronizes with b" sounds similar to me but it really is not.
  Bhakta: I did not see what C++ did with carries_dependency. Did they remove it?
  Gustedt: I think they kept it for backwards compatibility.
  Ballman: The latest working draft for C++ no longer has carries_dependency.
  Bhakta: If we could have in the agenda 4-5 papers that we would prioritize if we have more time, that would be great.
  Seacord: Who has not read this paper?: *8+4* **Myers** says: Once the agenda is full, there is 10 "if-time" papers, hold off on the fire hose.
  Bazley: I second **Bachmann**. I do not appreciate breaking earlier than Friday.
  Liber: There were more "in-time" papers than agenda papers.
  Uecker: Now it is a good use of time for paper feedback even if we do not vote on it.
  Straw Poll: Would WG14 like to adopt N3607 into C2y subject to possible objections before the next meeting? 1-0-1 + 23-0-3 = 24-0-4: strong consensus

- **Douglas**, Variable length prefixed length strings [N3608] (0.5 hours)

  Krause: This looks like a cool hack, like the whole UTF encoding. Have you considered language-independent representation of these strings?
  Douglas: That would suffer from the XKCD n+1 problem.
  Kaye: The length encoding is clever. Being invalid UTF8 is important.
  Colomar: This proposal can only get accepted over my dead body. System calls to the kernel take NTBS. It is standardized in C and POSIX. NginX had strong memory consumption requirements and never worried that size_t bytes were a problem. This is invalid UTF8, but there is existing code that assumes a string is valid UTF8.
  Douglas: You would have to cast. Not all OS's use NTBS's.
  Liber: Has this been adopted anywhere?
  Douglas: No, it is very much my invention.
  Bazley: You said my paper is orthogonal to your proposal, but that misses the point of my proposal. The real issue of C strings is a failure to extract the irrelevant.
  McCall: I am curious as to why you did not use a pointer indicator?
  Douglas: I refer you to the way LLVM stores things internally.
  Crammer: In other languages you have to deal with the C ABI; you must convert those strings to C NTBS's. What is your take on how that API problem should be handled for these types of strings?

Douglas: Everyone does strings slightly differently. My answer is yes, this would make it worse. But we are at the base of the tree and we can strengthen the whole tree.
Myers: This could be a self-contained API for CRFI.
Bachmann: This is very much special-purpose and not generally usable. We should not vote it in because there is almost no prior use of something out of the mainline.
Tarditi: Some people conjecture that size_t is too expensive. I do not agree. Systems with huge amounts of string processing, like Java or C# are fine. With regard to the fact that NTBS's keep biting people, we should upgrade the type system specifically for NTBS.
Kaye: What do you think of making these strings null-terminated for compatibility?
Douglas: I am not opposed to the idea. I would generalize treatment of arrays in general. The next thing would be string support.
Bazley: I agree. If we adopt a new string type, it should be opaque.
Celeste: You could always vote this in to CRFI.

- **Colomar**, Add operators _Minof and _Maxof [N3628] (0.5 hours)

  Celeste: The prefix operators seems wrong. I prefer library functions. The problem with a library function is that these take a type operand like sizeof() does.
  Colomar: That is done mainly for consistency with sizeof(). Also these are constant expressions, they return the same type as the input and you cannot do that with functions.
  Krause: The natural thing to do is macros in the standard library. Or you can use _Generic. The macro can evaluate to a builtin.
  Myers: I believe this should be a language feature, not a library feature. The same goes for offsetof().
  Ballman: As a language feature, this is expensive. I am not sure this is worth the case.
  Mailhol: I support this. I prefer names of type_max/type_min.
  Gustedt: I agree with **Ballman** that more interesting primitives would be to get the widths of the type. But the return type here is simple.
  Celeste: That is what I would like to see. My one concern is that these types of operators sort of preclude a general type traits library.
  Colomar: I have several use cases such as a strtol() variant. Also, the operators all end in "of", so you know _minof() is an operator.
  Bazley: I support this paper.
  Łukasiewicz: I support this paper. The library may be independent of the compiler.
  Straw Poll: Would WG14 like to adopt N3628 into C2y subject to possible objections before the next meeting? 3-1-0 + 10-4-7 = 13-5-7: do not have consensus

Afternoon break

- **Colomar**, Allow calling static inline within extern inline [N3622] (0.5 hours)

  Ballman: The compiler can see all definitions so it should have no troubles. But calls to calls to calls should require some static analysis. As a constraint we must issue a diagnostic. Removing the constraint requires quality of implementation to catch issues.
  Colomar: If your function calls some other function, it should know where it is from.
  Myers: If you remove constraints, you also need to remove the example 2 in section 6.7.5, paragraph 13.
  Bazley: I do not believe any developer would thank us for putting this constraint in place. This is an artificial unnecessary barrier between two camps of C developers. The constraint is not sufficiently motivated.
  Celeste: From implementation experience, it was not reasonable to expect static analysis.
  McCall: I am confused between extern and inline.
  Gustedt: I am in favor of that, because Clang repeatedly warns me about functions in their header file; they have inline substituted.
  Straw Poll: Would WG14 like to adopt N3622 into C2y subject to removing the second example in 6.7.5p13 and subject to possible objections before the next meeting? 2-0-0 + 16-1-4 = 18-1-4: strong consensus

- **Colomar**, Refactor syntax of directives [N3632] (0.5 hours)

  Kaye: I find this format difficult to read.
  Colomar: "def" is a multi-line macro definition. If we make this change, it should be conditional on WG21 also making this change.
  Celeste: This cleanup makes the document more useful. The preprocessing is ad hoc. Cleanups need great care to make sure they break nothing.
  Dusíková: I like the changes.
  Gustedt: I cannot vote for this paper, because it is in a preliminary state. The rationale says nothing about what it does. The proposal is wording that is difficult to parse. It needs examples of old vs. new syntax.
  McCall: This is too technical a paper to be viewing in this section.
  Uecker: How does the preprocessors in C and C++ on semantics? Is there already a lot of divergence?
  Ballman: Not identical but pretty close to it. Usually WG21 adopts changes applied by WG14.
  Gusted: I have a project for a language-independent preprocessor.
  Celeste: Preprocessors do have slight different wording, mainly due to other differences in the standards.
  Opinion Poll: Would WG14 like to adopt something along the lines of N3632 into C2y? 2-0-0 + 7-3-14 = 9-3-14: no direction.
  Svoboda: I could not read the normative text or understand the rationale, so I could not vote yes or no.
  Bhakta: I am abstaining from all these "Other Business" papers because I did not prepare for them.

- **Colomar**, (potentially) reserve stp*(), wcp*(), memp*(), wmemp*() [N3614] (0.5 hours)

  Kaye: Are these names not being standardized by POSIX?
  Krause: I am skeptical about the proposals that these prefixes appear in. "stp" and "wcp" are prefixes not traditionally reserved. Do we have statistics on how common they are?
  Ballman: They should be zero. At some point we might use these prefixes.
  Bazley: These API's are not necessary yet, they do not yet exist in ISO C. The motivation is weak; based just on functions **Colomar** uses. I would not assume "stp" is a string-based prefix.
  Liber: I am wholly against reserving names in advance. The public does not care.
  Kaye: There are other functions that return a pointer to a string that do not use any of these prefixes.
  Banham: I do not see the point of reserving stuff that is not in the standard.
  Svoboda: I suggest reviewing this document after we review and accept the other documents that define functions with these prefixes.
  Colomar: I would suggest voting on two papers at once.

- **Colomar**, add stpcpy(3), wcpcpy(3), mempcpy(3), wmempcpy(3) [N3662] (0.5 hours)

  Krause: This provides very little new functionality. They only differ in return values. I am opposed.
  Gustedt: If you want to introduce functions, please describe what they do, and motivate us as to why you want them in.
  Myers: This was submitted after the deadline, so I object to voting on it but we can discuss it.
  Bazley: I like that you base things on the charter, but I disagree with the reasoning. I am not convinced these functions enable secure programming. If we are going to put new functions in, they should be more useful than these.
  Seacord: Could we do a directional poll?
  Colomar: The only change in mempcpy() is the return value, but stpcpy() is not a drop-in replacement.
  Opinion Poll: Would WG14 like something along the lines N3662 and N3661 into C2y? 2-1-2 + 3-13-8 = 5-14-10: clear direction not to do this

- **Colomar**, add memeq(), wmemeq() [N3617] (0.5 hours)

  Krause: It is too close to memcmp(), and not a sufficient improvement.
  Bazley: I was in favor of streq(), but not in favor of this, because I do not use memcmp().

- **Colomar**, add [w]memzero(), [w]memzero_explicit() [N3619] (0.5 hours)

Seacord: One of our principles is not to duplicate mechanisms. I like a CRFI library especially if there are multiple overlapping string or memory functions.
Douglas: There was memset_explicit().
Bazley: I feel similar for multi-line macros. But I recognize that is a bad thought. I try not to use memset(), I prefer assignment.
Crammer: These are nice features we might want to use, but they are never justified. Why does the standard library need it?
Kaye: I am guessing that there would be minimal implementation effort. It makes intent explicit and makes it harder to misuse.
Celeste: Why are these const pointers?
Colomar: We need memcpy() and memset_explicit(). The rest of the string library is superfluous.
Ballman: This has the exact same interface as bzero(), which was deprecated and removed.
Colomar: They are not the same; this returns a pointer. I do not know why bzero() was deprecated. 30 years ago many bzero() implementations were wrong.
Krause: We have memset(), it provides more functionality than memzero(), and one less character in the function name, and it is commonly used. I do not think memzero() provides enough additional benefit.
Bazley: You do not need to use memset(). Compiler authors started recognizing loops and replaced them with memset(). There are arguments that memset() are too simplistic.
Crammer: One reason to have things in the standard library is compiler primitives which do not need to be language-level features, memset() being one such example. Other examples include popcount().
Opinion Poll: Would WG14 like something along the lines of N3619 with all the "const"s removed? 2-2-2 + 3-14-7 = 5-16-9: clear direction not to do this

Friday, 29 August

- **Múgica**, Discarded, IV [N3549] (0.5 hours)

Krause: The wording is based on N3546 rather than the latest C2y draft. Only proposal 2 is based on the bool paper, proposal 1 is not.
Bhakta: The first proposal takes the premise that some expressions are constant expressions. This premise is used throughout the proposal.
Myers: This is an example, the wording is wrong. This is what you get if you take a literal but nonsensical reading of the standard.
Bhakta: In the section 6.5.4, paragraph 3 change, the change seems to imply that there is no integer constant expression for alignof. This is not what it should be.
Opinion Poll: Would WG14 like something along the lines of Proposal 1 of N3549? 4-0-1 + 7-1-5 = 11-1-6: direction in favor
Krause: Proposal 2 looks too permissive.
Bhakta: If this depends on a paper that did not make it, should we still consider this?
Seacord: We are just getting feedback.
Banham: I am puzzled by statements that arrays cannot have negative indices.
Bhakta: We made that a constraint violation.
McCall: I like the C++ approach that constant expressions cannot have undefined behavior. This feels redundant.
Bazley: A paper that proposes two alternatives is not sufficient.
Bhakta: Regarding his two final questions, my answers are: No, no, because the as-if rule already allows it.
Celeste: This ties in to what is and is not an integer constant expression.
Opinion Poll: Does WG14 wish to allow implementations to make the ||, && and conditional operators discard the second (or third) operand when the first one is a constant expression other than an integer constant expression? 1-4-0 + 2-6-7 = 3-10-7: direction against
Opinion Poll: Does WG14 wish to allow implementations to make some operators discard an operand when it can be determined that the operand will never be executed in the abstract machine? 0-4-0 + 0-11-4 = 0-15-4: clear direction against

- **Múgica**, What does “to evaluate a type name” mean? [N3548] (0.5 hours)

  Seacord: "Disappears" is not a standard term.
  Bhakta: It does not seem like anyone gets this wrong. I'm not sure there is a problem here.
  Svoboda: Is the type name evaluated at runtime? The "at runtime" phrase is confusing, it raises more questions than it answers.
  Uecker: It can be, though not for types in sizeof expressions.
  Bhakta: People are ignoring the fact that adding text adds cost.
  Kaye: Is there consent about adding something to the standard? Could we discuss this at the start of the next meeting?
  Opinion Poll: Would WG14 like something along the lines of N3548? 4-0-0 + 5-4-7 = 9-4-7: direction

- **Múgica**, Generic replacement (v. 2 of quasi-literals) [N3605] (0.5 hours)

  Myers: Is this more about comparing literal constants of a type?
  Celeste: I like the direction. The parentheses is a bigger motivator. I prefer this to be defined as locally to _Generic expressions.
  Bhakta: The nodiscard case is accepted by Clang without issue, in contrast to the static_assert case.
  Straw Poll: Would WG14 like to adopt N3605 into C2y? 3-0-1 + 7-0-8 = 10-0-9: strong consensus to accept while requesting editorial changes
  Seacord: Some people thought this was an along-the-lines-of poll. We will re-poll.
  Straw Poll: Would WG14 like to adopt N3605 with editorial changes in reflector message 33542 into C2y? 2-0-1 + 7-1-8 = 9-1-9: not strong consensus.

- **Múgica**, What are the operands of Generic [N3561] (0.5 hours)

  Svoboda: I prefer alternative 2, since _Generic expressions depend on both the type name.
  Celeste: …if the intro suggests no difference.
  Ballman: Does this change any behavior?
  Uecker: No, it just clarifies wording.
  Bazley: **Uecker**, which alternative do you prefer?
  Uecker: I prefer alternative 2, but for no particular reason.
  McCall: Did we not accept some of the discarded changes?
  Uecker: It is related because the discarded changes use the term "operand".
  Bhakta: Did we decide to create a Description section or keep it under Semantics?
  Opinion Poll: Would WG14 prefer Alternative 2 over Alternative 1 in N3561? 2-0-1 + 5-5-2 = 7-5-3: weak preference for Alternative 2

- Revisit **Lenga**, Additional String comparison functions to complement strcmp() V2 [N3567] (0.5 hours)

  Seacord: I agree with that.
  Bazley: I agree. The design for Optional is based on the premise that most C pointers should not be NULL. NULL has two meanings: 1. Something went wrong. 2. I am not providing an object.
  Celeste: I have no strong feeling on the prefix. We want to bring in a minimum of undefined behavior. Allowing new string functions to have undefined behavior on NULL introduces a new demon.
  Krause: I agree. It does make a performance difference to handle NULLs.
  Bazley: I am sympathetic to defensive programming. But it does not scale.
  Colomar: My feelings on this mirror my feelings on Annex K. In theory it removes undefined behavior. But it introduces second-order bugs that trigger more undefined behavior. I prefer having undefined behavior as early as possible.
  Myers: This is not a new undefined behavior.

7. Other Business

Time permitting:

- **Celeste**, `auto` as a placeholder type specifier, v3 [N3579] (0.5 hours)
- **Johnson**, Namespaces Without Name Mangling [N3491] (0.5 hours)
- **Johnson**, Generic Data Structures Without Name Mangling [N3520] (0.5 hours)
- **Lenga**, Additional String comparison functions to complement strcmp() V2 [N3567] (0.5 hours)
- **Colomar**, add strpfx(), stppfx(), wcspfx(), and wcppfx() [N3612] (0.5 hours)
- **Colomar**, add strsfx(), stpsfx(), wcssfx(), wcpsfx() [N3613] (0.5 hours)
- **Celeste**, Remove the imaginary I, v2 [N3581] (0.5 hours)
- **Celeste**, `if` declarations, v5.1, wording improvements [N3580] (0.5 hours)
- **Ornato**, Declaration-level static assertions [N3641] (0.5 hours)
- **Łukasiewicz**, Clarify terminology for obsolete features [N3642] (0.5 hours)
- **Adams**, Literal functions [N3645] (0.5 hours)
- **Thomas**, Clarification of range error condition for atan2v [N3646] (0.25 hours)
- **Thomas**, Semantic rules for constant evaluation [N3647] (0.25 hours)
- **Thomas**, Improved language for return type vs. return value [N3648] (0.25 hours)
- **Krause**, Named Address Space Type Qualifiers for C2y [N3651] (0.5 hours)
- **Uecker**, Accessing the Context of Nested Functions [N3654] (0.5 hours)
- **Banham**, Make implicit undefined behavior in mtx_destroy() explicit [N3655] (0.5 hours)
- **Meneide**, Functions with Data: Closures in C [N3657] (0.5 hours)
- **Celeste**, Simplified lexical scope for labels, v2 [N3658] (0.5 hours)
- **Belov**, Considering expressions based on restrict pointers as pure rvalue expressions [N3659] (0.5 hours)
- **Colomar**, add a malloc(3)-based sprintf(3) variant [N3660] (0.5 hours)
- **Colomar**, Standard prefixed attributes [N3661] (0.5 hours)
- **Colomar**, add stpspn(), stpcspn(), wcpspn(), wcpcspn() [N3663] (0.5 hours)
- **Colomar**, add strchrcnt(), strchrscnt(), wcschrcnt(), wcschrscnt() [N3664] (0.5 hours)
- **Colomar**, add st*rspn(), st*rcspn(), wc*rspn(), wc*rcspn() [N3665] (0.5 hours)
- **Colomar**, add stpsep(), wcpsep() [N3666] (0.5 hours)
- **Colomar**, add strsep(), wcssep() [N3667] (0.5 hours)
- **Colomar**, Split formatted I/O sections (Editorial) [N3668] (0.5 hours)
- **Colomar**, Split memory management section (Editorial) [N3669] (0.5 hours)
- **Colomar**, Rename s/strpbrk/strchrs/ [N3670] (0.5 hours)
- **Colomar**, Subdivide string API sections [N3671] (0.5 hours)
- **Gustedt**, Properly specify the interaction of library calls for mutexes [N3672] (0.5 hours)
- **Gustedt**, Properly specify the interaction of library calls for condition variables [N3673] (0.5 hours)
- **Svoboda**, WG14's C indentation and brace styles [N3650] (0.5 hours)
- **Svoboda**, Educational Undefined Behavior [N3534] (0.5 hours)
- **Bazley**, Enhanced type variance (v2) [N3674] (0.5 hours)
- **Uecker**, Earthly Demon: Accessing a Member of an Atomic Structure or Union (Updates N3564) [N3624] (0.5 hours)

Scheduled for Spring 2026:

- **Colomar**, Add directives #def and #enddef (updates N3524) [N3531] (0.5 hours)
- **Colomar**, sizeof(void) == 1 [N3522] (0.5 hours)
- **Garcia**, typeof(return) [N3454] (0.5 hours)

Old papers that haven't been scheduled yet pending request from the author:

- **Bachmann**, Make pointer type casting useful without negatively impacting performance - updates n2484 [N2658]
- **Grüninger**, Add min, max for integers to C [N3160]

8. Recommendations and Decisions Reached

8.1 Review of Decisions Reached

Straw Poll: Would WG14 like to submit TS 25007 to SC22 for balloting? 13-0-8 + 0-3-4 = 13-3-11: consensus
Straw Poll: Would WG14 like to adopt N3348 as is into C2y? 13-6-6 + 3-0-2 = 16-6-8: consensus
Straw Poll: Would WG14 like to adopt N3457 into C2y without the recommended practice first line and two bullets, and 2147483648 as the limit? 24-0-2 + 3-0-1 = 27-0-3: consensus
Straw Poll: Would WG14 like to adopt N3558 as is into C2y? 4-0-0 + 15-0-6 0 = 19-0-6: consensus
Straw Poll: Would WG14 like to accept the suggested correction for issue 1002 from N3609 into C2y and into previous versions of C? 4-0-0 + 14-0-3 = 17-0-3: consensus
Straw Poll: Would WG14 like to accept the suggested correction for issue 1003 from N3609 into C2y and previous versions of C? 1-0-2 + 16-0-4 = 17-0-6: consensus
Straw Poll: Would WG14 like to accept the suggested correction for issue 1005 from N3609 into C2y? 0-0-4 + 12-0-9 = 12-0-13: weak consensus, adopted
Straw Poll: Would WG14 like to leave the standard as is in issue 1007 from N3609? 2-0-2 + 8-11-0 = 10-11-2: no consensus
Straw Poll: Would WG14 like to have a constraint violation in issue 1007 from N3609? 1-1-2 + 8-10-3 = 9-11-5: no consensus
Straw Poll: Would WG14 like to change the behavior to reflect C++ in issue 1007 from N3609? 3-0-1 + 6-9-6 = 9-9-7: no consensus
Straw Poll: Would WG14 like to accept the suggested correction to Issue 1008 in N3609? Adopted by unanimous consent
Straw Poll: Would WG14 like to accept the suggested correction to Issue 1009 in N3609? Adopted by unanimous consent
Straw Poll: Would WG14 like to accept the suggested correction to Issue 1011 in N3609? Adopted by unanimous consent
Straw Poll: Would WG14 like to adopt N3559 as is into C2y? 2-0-1 15-0-5 = 17-0-6: consensus
Straw Poll: Would WG14 like to adopt N3559 to obsolete versions of C? 2-0-3 + 17-1-3 = 19-1-6: consensus
Straw Poll: Would WG14 like to adopt N3602 as is into C2y? 2-3-1 + 7-7-11 = 9-10-12: no consensus
Straw Poll: Would WG14 like to adopt N3545 as is into C2y? 1-1-2 + 2-15-9 = 3-16-11: no consensus
Straw Poll: Would WG14 like to accept the wording in reflector message 30750 for future editions of TS 18661-4? 1-0-2 + 13-0-9 = 14-0-11: strong consensus
Straw Poll: Would WG14 like to accept N3557 as is into C2y? 2-0-2 + 11-6-7 = 13-6-9: not strong consensus, not adopted.
Straw Poll: Would WG14 like to adopt N3577 as is into C2y? 3-0-0 + 21-1-5 = 24-1-5: strong consensus, adopted
Straw Poll: Would WG14 like to accept the suggested correction in N3511? Adopted by unanimous consent
Straw Poll: Would WG14 like to adopt N3611 as is into C2y? 2-1-1 + 8-15-6 = 10-16-7: no consensus
Straw Poll: Are there any objections to adopt the suggested wording in issue 1016 from N3609 for C2y? *none*
Straw Poll: Are there any objections to adopt the suggested wording in issue 1016 from N3609 for obsolete versions of the C standard? *none*
Straw Poll: Would WG14 like to adopt N3623 with the wording change into C2y? 2-0-1 + 15-3-6 = 17-3-7: strong consensus
Straw Poll: Would WG14 like to accept N3563 into C2y? Adopted by unanimous consent
Straw Poll: Would WG14 like to accept N3532 into C2y? Adopted by unanimous consent
Straw Poll: Any objection to adopting N3532 into previous versions of the standard? Adopted by unanimous consent
Straw Poll: Would WG14 like to adopt N3525 as is into C2y? 2-0-2 + 16-3-4 = 18-3-6: adopted
Straw Poll: Would WG14 like to adopt N3485 with the editorial correction into C2y? 1-2-0 + 18-5-5 = 19-7-5: weak consensus, not adopted
Straw Poll: Would WG14 like to adopt N3621 into C2y? 3-2-0 + 6-11-6 = 9-13-6: no consensus
Straw Poll: Would WG14 like to accept N3544 into C2y? Adopted by unanimous consent

Straw Poll: Would WG14 like to accept N3535? Adopted by unanimous consent
Straw Poll: Any objection to adopting N3536 into the standard? Adopted by unanimous consent
Straw Poll: Any objection to adopting N3537 into the standard? Adopted by unanimous consent
Straw Poll: Any objection to adopting N3500 into the standard? Adopted by unanimous consent
Straw Poll: Any objection to adopting N3598 into the standard? Adopted by unanimous consent
Straw Poll: Would WG14 like to adopt N3607 into C2y subject to possible objections before the next meeting? 1-0-1 + 23-0-3 = 24-0-4: strong consensus
Straw Poll: Would WG14 like to adopt N3628 into C2y subject to possible objections before the next meeting? 3-1-0 + 10-4-7 = 13-5-7: do not have consensus
Straw Poll: Would WG14 like to adopt N3622 into C2y subject to removing the second example in 6.7.5p13 and subject to possible objections before the next meeting? 2-0-0 + 16-1-4 = 18-1-4: strong consensus
Straw Poll: Would WG14 like to adopt N3561 into C2y? 3-0-1 + 7-0-8 = 10-0-9: strong consensus to accept while requesting editorial changes
Straw Poll: Would WG14 like to adopt N3605 with editorial changes in reflector message 33542 into C2y? 2-0-1 + 7-1-8 = 9-1-9: not strong consensus.

8.2 Review of Action Items

Action Item: **Seacord**: Submit TS 25007 to SC22 for balloting.
Action Item: **Ballman**: Apply N3532 to the obsolete versions of the standard.
Action Item: **Seacord**: Work with Simonsen to update the WG14 reflector.
Action Item: **Celeste**: Contact **Bhakta** about the Canada (Fall 2026) meeting.
Action Item: **Douglas**: Check with POSIX for direction for issue 1004 in N3609.
Action Item: **Bhakta**: CFP to submit a paper as per reflector message 30480.

8.3 Review of Online Votes

Does WG14 want to add N3411 to obsolete versions of C? 14-1-3: adopted
Does WG14 want to add N3517 into C2y? 13-0-5: consensus adopted
Does WG14 want to add N3652 into C2y? 19-0-2: consensus adopted

8.4 Re-approve Agenda

Motion to approve the agenda (N3680). **Celeste** moves. **Svoboda** seconds.
Mailhol: Do we have a list of papers to vote online?
Seacord: N3581 and N3580 will be voted online.
Banham: Can we keep the ISO invitations more up-to-date?
Seacord: I did notify the reflector.
Svoboda: The agenda has evolved since the meeting started. Do we approve the original agenda or the current agenda (what we actually did)? In my opinion, the latter agenda is redundant, it is just the outline of the minutes.
Objections to approving the agenda? *None*

9. Thanks to Host

10. Adjournment

Svoboda motions to adjourn. **Uecker** seconds. Objections? *None*

End