# Final Minutes for 10 - 14 June, 2024

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

WG14 / n3373

### Dates and Times

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

| | |
|---|---|
| 10 June, 2024 | 13:30 – 17:00 UTC |
| 11 June, 2024 | 13:30 – 17:00 UTC |
| 12 June, 2024 | 13:30 – 17:00 UTC |
| 13 June, 2024 | 13:30 – 17:00 UTC |
| 14 June, 2024 | 13:30 – 17:00 UTC |

### Meeting Location

This meeting is virtual via Zoom.

### Meeting information

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

### Local contact information

Robert Seacord <rcseacord@gmail.com>

## 1. Opening Activities

### 1.1 Opening Comments (Seacord)

Welcome to 70th meeting.
Seacord likes to keep everything on schedule. Leave time for straw polls - at least 5 minutes.
Wait till section of agenda to hear from national bodies.

### 1.2 Introduction of Participants/Roll Call

| Name | Organization | NB | Notes |
|---|---|---|---|
| Aaron Bachmann | Efkon GmbH | Austria | Austria NB |
| Aaron Ballman | Intel | USA | |
| Alex Celeste | Perforce Software | USA | MISRA |

| Name | Organization | NB | Notes |
|---|---|---|---|
| Ash Chronister | Apple | USA | |
| Chris Anley | NCC Group | USA | |
| Chris Bazley | ARM | UK | |
| Clive Pygott | LDRA Technology | USA | |
| Corentin Jabot | Freelance | France | Guest |
| David Svoboda | SEI/CERT/CMU | USA | Undef. Behav. SG Chair |
| Douglas Teeple | Plum Hall | USA | |
| Eskil Steenberg | Quel Solaar | Sweden | Sweden NB |
| Fred Tydeman | Keaton Consulting | USA | INCITS/C Vice Chair |
| Freek Wiedijk | Plum Hall | USA | |
| Henry Kleynhans | Bloomberg | USA | |
| Jakub Łukasiewicz | Motorola Solutions Systems Polska | Poland | Poland NB |
| JeanHeyd Meneide | NEN | Netherlands | Neth. NB; C++ Comp. SG |
| Jens Gustedt | INRIA/ICube | France | France NB |
| Joseph Myers | Red Hat | UK | UK NB |
| Joshua Cranmer | Intel | USA | |
| Jyoti Kushwaha | Bureau of Indian Standards | India | India NB |
| Martin Uecker | Graz University of Technology | Austria | |
| Michael Wong | Codeplay | Canada, UK | WG21 Liaison |
| Niall Douglas | Ireland Ned Productions Ltd | Ireland | Ireland NB |
| Nick Stoughton | Logitech | USA | SC22 Austin Group Liaison |
| Nikita Popov | Red Hat | Germany | Guest |
| Patrizia Kaye | BSI | UK | |
| Paul McKenney | Meta/Facebook | USA | |
| Peter Sewell | University of Cambridge | UK | Memory Model SG Chair |
| Philipp Krause | Self | Germany | |
| Rajan Bhakta | IBM | USA, Canada | INCITS/C Chair |
| Robert Seacord | Woven Planet North America Inc | USA | |
| Roberto Bagnara | BUGSENG | Italy | Italy NB, MISRA Liaison |
| Ted Johnson | Hewlett Packard | USA | |
| Yeoul Na | Apple | USA | |

*Seacord*: Would the first-time attendees please un-mute and introduce themselves?

*Kaye*: I am Patrizia Kaye from BSI. I've been using C for 15-20 years. C is neat!
*Anley*: I am Chris Anley, chief scientist at NCC group. My interest is in the security of C, and practices that help folks write software. Experience with C goes back 30 years. C is important!
Seacord: I guess.
*Kushwaha*: I am involved with the National Standards bodies for India. I have experience of working with C and have dabbled in C++. So it was a logical step for me to join WG14.
*Na*: I am Yeoul Na from Apple. I am working on bounds safety for the C language. I am also working on up-streaming the Clang bounds safety extension. I would like to get feedback from the committee and potentially open-source language model.
*Chronister*: I am Ash Chronister, also from Apple. I am here out of general interest and to support various proposals.
*Seacord*: Did someone make you come?
*Chronister*: No, my idea.
*Seacord*: Someone from C++ committee said they'd send someone from Apple.
*Na*: That could have been me.

### 1.3 Procedures for this Meeting (Seacord)

*Seacord*: We're going to discuss a bunch of papers. Some are more ready than others so we'll take straw polls. There are two types of polls: 1. to add something to the C2y standard. These need consensus, more than a majority but less than everyone. Everyone attending gets to vote, we don't break down votes by company or national body. The 2nd type of poll is directional when someone wants to decide whether they are heading in the right direction regarding a paper. We're very comfortable giving people work to do; it doesn't mean a proposal will be accepted. We are an interesting group. We are not an easy group but rather a useful learning experience.
*Bhakta*: Anyone can speak. You don't have to hold back if you're new.
*Seacord*: Raise your hand to speak. We will turn off the queue as we near the end of the time for a paper.
*Bhakta*: What is the chat policy?
*Seacord*: My predecessor didn't like people writing in the chat but I don't mind. But pay attention to the audio. Sometimes it is easier to paste a link to a document in the chat rather than speak it.
*Seacord*: I am going to have *Bazley* as scribe. (Minutes were taken by *Bazley*, *Svoboda*, and *McKenney*.)

### 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 [N 2613]

### 1.5 Approval of Previous Minutes

WG 14 Minutes, January 2024 [N3227] (motion)

Minutes can be finalized with changes to correct typo and Joseph's employer.
*Svoboda*: I move to approve minutes.
*Ballman*: Seconded

*Seacord*: Are we missing the final minutes from October?
*Pygott*: I will ask for a document number and then send it to be posted.
*Seacord*: That would be great.

### 1.6 Review of Action Items and Resolutions

- *Seacord*: submit TS18661-5 for ballot

  *Seacord*: Done

- N3192: making hexdigits sequential

  *Seacord*: That has been added to a document that's only in my head. *Celeste*, did you have a chance to look at the latest working draft of the standard?
  *Celeste*: I've made no change to the relevant area.
  *Ballman*: no objections from Clang

- *Celeste*: Submit a new work item proposal for defer TS

  *Seacord*: Which TS?

  *Celeste*: It doesn't have a number because it gets that out of the process.
  *Seacord*: Anything else?
  *Celeste*: All other decisions were comment resolution.

**1.7 Approval of Agenda [n3213] (motion)**

*Bhakta*: Moved
*Stoughton*: Seconded

**1.8 Identify National Bodies Sending Experts**

*Seacord*: Trying to find one person who would be head of delegation for US. That would be you, Rajan, for US.

| Person | Nation |
|---|---|
| Jens Gustedt | France |
| Jyoti Kushwaha | India |
| Joseph Myers | UK |
| Jakob Łukasiewicz | Poland |
| Michael Wong | Canada |
| Roberto Bagnara | Italy |
| Martin Uecker | Austria |
| Niall Douglas | Ireland |
| Eskil Steenberg | Sweden |
| Philipp Krause | Germany |
| JeanHeyd Meneide | Netherlands |

*Krause*: I am not a member of the national standards body but they agreed to participation.
*Seacord*: if I had to guess, you do not represent the national body if you are not yet a member. Never had more national bodies in a meeting than today. That's impressive. We just need 5 votes from member nations to create a work item or something like that.

### 1.9 Note where we are in the current C23 schedule [N 3156]

*Seacord*: Here's the C23 schedule. We are trying to get published before the standard is automatically canceled. Nothing most people on the committee can do about that at the moment. We keep submitting versions of the standard and the ISO editor keeps rejecting it. Hopefully we are iterating towards agreement. I heard from David Keaton what a pain it is and am now experiencing it first-hand. Mostly a pain for *Meneide*.
*Myers*: Is there a way of getting issues sorted out at CD time?
*Seacord*: I don't know that there is. There Might be specific comments we're getting back from the editors. Would like diagnostics for the whole document.
*Wong*: I can give a concrete understanding of the problem; it has been affecting C++23. In the past we could get a TS through in one iteration with some suggestion. Now we can't get through with four iterations. I consider myself experienced in doing this properly. Two major things: the pickiness of the editors is enormous. CD suggestions probably won't work. They won't allow amendments to existing ISO documents, which we do a lot. And we can't change existing clauses.
Herb and Bill are joining forces to send a complaint to ISO. This is a human cost to the editors, that is, they "don't want to do this anymore." Maybe ISO needs to back off on their pickiness. Maybe they were told to be more picky.
*Seacord*: We would like to join in with complaints. Editors have their own careers.
*Ballman*: We rejected several things with good rationale. I would like to know why we wasted committee time disposing of comments that they disregard.
*Seacord*: The first time though the process, we thought they would listen to us. We had a recent version of the draft and all the paragraph numbers were gone. That was horrible: really hard to read.
*Celeste*: In the absence of *Meneide*, the last time we chatted, he said they had backed down on some of the big changes. It sounds like the extreme changes have been withdrawn.
*Wiedijk*: I have been looking at the last requirements of ISO with *Meneide* and put in some changes. The only thing left is that they don't like the use of "may." I have been making a huge list of "may"s and changing them. *Meneide* would have liked to have had a new iteration before the meeting.

*Wiedijk*: There are several places where "may" is used in the sense of "possible" instead of "allowed", but it is ambiguous whether the implementation or the user is allowed to do something.
*Bhakta*: My suggestions took out the word "may" so you don't have these problems.
*Wiedijk*: A lot of "may"s – about 300 in the standard. If they don't add new stuff, it should be converging.
*Seacord*: *Bhakta*, please email your suggestions to *Wiedijk* and *Meneide* in case they don't have them to resolve this quicker.
*Bhakta*: That was sent as part of editorial review.
*Seacord*: Send them again.

*Meneide*: I am trying to get C23 through ISO, we have made several drafts to conform with ISO requirements. We got an exception to keep italics in inline definitions. and some other exceptions for other things we need. Substituting "may" with "can" was the last thing they requested, it is done. We will submit again. (This is a big ping-pong game between *Meneide* and *Wiedijk* vs. ISO team)
*Gustedt*: Lots of difficulties with ISO. The idea was that this should come from several national bodies. If any other national bodies have an idea to join us in effort in improving this process/relationship, please contact me

## 2. Reports on Liaison and Collaboration Activities

### 2.1 ISO, IEC, JTC 1, SC 22

### 2.2 National bodies in WG 14

### 2.3 WG 21

### 2.4 WG 23

*Pygott*: This used to be me but I've resigned so it's not me anymore.
*Seacord*: Congrats. It might be *Wong* as well.
*Pygott*: No, that was WG21's SC23, not WG23.
*Wong*: The St. Louis meeting is in 2 weeks. There is an administration call in 1 hour. I will try to merge any TS. The next meeting, in Poland is the last meeting for C++26 features we haven't seen before. The major things are contracts, safety and security.
*Seacord*: Poland is where people submit crappy proposals to get their foot in the door.

### 2.5 MISRA C

*Celeste*: A new MISRA C edition aims to be out next year. We are aiming to expand into POSIX and GNU C. It is in early draft state.
*Seacord*: Are there any updates about me joining the working group?
*Bhakta*: Does GNU C have a document that you're basing off?
*Celeste*: Yes, using the GNU C manual. Instead of "Do not use extensions" but we want to say, "Do not use computed goto targets" with different severity.

### 2.6 Austin Group

*Stoughton*: POSIX23 is an official standard in the open group. It is in the final stages of DIS ballot. I sincerely hope there are no comments. It is based on C17. We are starting to consider what goes into the next edition but these are the very early days.

### 2.7 Unicode Consortium

*Seacord*: Unicode Consortium not is not here. Last call on WG23 – none heard.

### 2.8 Other Liaison or Collaboration Activities

## 3. Study Groups

### 3.1 C Floating Point Study Group activity report (Rajan)

*Bhakta*: If you have any issues relating to C2y, please bring them forward. TS parts 4 and 5 (TS 18661-4 and TS18661-5) should be good to publish when we have C23.

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

*Sewell*: ISO have come back with many complaints, some of which are stupid, some of which would break the text. I will talk to *Wiedijk* and *Meneide* about what they've been able to push back in.
*Kleynhans*: I am hoping to get changes done so we can resubmit, and hoping to get some feedback before we do that.

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

*Ballman*: The Compatibility study group has become a bit more active, all currently happening on the reflectors. No meetings yet.

### 3.4 Undefined Behavior Study Group activity report (Svoboda)

*Svoboda*: We have three documents coming up. First is writing to multibyte char files. Bigger one is still to be submitted – educational TS, current status quo of Undefined Behavior. *Steenberg* is in charge of that. I guess submit it in a month.
*Steenberg*: *Bazley* and I will have a call and he will give input. I am happy to get input from others. I will submit when there is no more input.
*Seacord*: For new folks, contact study group chair if you are interested in joining one of these study groups

### 3.5 New Study Group proposal (if applicable)

### 3.6 Infrastructure Group (Meneide)

*Ballman*: The WG21 link is a re-director. For the core issues list and the libraries list. For the last 5 years there was a domain squat but they messed up and forgot to renew it so I snagged it out from under them. I am looking for someone with web programming experience to provide the same service for WG14. I don't have time to improve my web-programming skills to do it.
*Seacord*: My son-in-law might be able to volunteer.
*Bhakta*: I posted to the C reflector (message 25766) to try to track consensus. I am not going to use it as a replacement because there should be human input. There are holes in that but it's a guideline. *Gustedt* already commented. *Meneide* is happy with having it as part of the infrastructure group. We are not planning on spending more time on it.
*Seacord*: I took your program and tried to compile it for WebASM. Both compilers hung after waiting two hours. I thought that would be a good idea. Stick them on a web page and use that as the user interface. We never really had any effort or thought about the relationship between C and WebASM. If we could promote C as a language for developing web sites then that would be a new market for us.
*Myers*: I have done work on converting past issues. Working reasonably well. I would still need to do a lot to convert HTML to Markdown for the tracker.
*Seacord*: I am afraid to ask *Meneide* about it. I wasn't planning for all the extra work getting the standard past the ISO editor. I prefer to leave him alone until the standard is published.

*Meneide*: No progress, working on the paper. I was supposed to have a paper out, but not done.

## 4. Future Meetings

### 4.1 Future Meeting Schedule

30 September - 04 October, 2024 Minneapolis, Minnesota Hybrid
Spring, 2025 Candidates: Graz, Austria; Spain, Bristol, UK

*Seacord*: 20th Sept in Minneapolis. *Celeste* is the host.
*Celeste*: The meeting is in the process of being planned. A venue info sheet was supposed to be ready for today. Approval for block hotel booking will hopefully be on the info sheet. Please email me directly as well as signing up on the ISO info sheet so touch badges are available so we don't need to hold the door. This is based on last time.

*Seacord*: For Spring 2025: I've had soft offers for Graz and for Spain. The idea for Bristol was that we could have this at the same time at the ACCU conference. The problem is that nobody would be able to attend both. I'm thinking of crossing it off for that reason. Thailand is interesting, but with the Downside in that it's a long flight. The plus side is a nice hotel for $70 a night. Is there anybody who wants to propose a new location? Doesn't sound like there is. Clearly we need to firm this up soon. If you've suggested a venue then please firm up. I think I'm at the point where I'll jump on the first firm offer.
*Gustedt*: Keep dates broad: Feb through April. If we keep it broad then that gives us more possibility. We had to have a meeting at CMU once during spring break because that's when the rooms became available. Pick a

location where we have the least transport for the group
*Seacord*: Taking a bus wasn't horrible.

ISO changed the rules for hybrid meetings. There is no longer any distinction between hybrid meeting and in-person meeting. In Strasbourg we had to stop the meeting whenever the connection went down. This happened multiple times although each was for only a short duration. In Minneapolis we'll continue with the agenda, so in-person participation is more important
*Ballman*: That seems reasonable but can we try our level best to have good connections and audio software? I remember pre-pandemic times and it doesn't work. Strasbourg worked great.
*Seacord*: I have confidence in *Celeste* and Minneapolis. I have less confidence with Thailand. We'll see. Maybe we'll end up in Graz.

### 4.2 Future Deadlines

Note: Please request document numbers by one week before these dates.
Pre-Virtual – 10 May, 2024
Pre-Minneapolis Hybrid 2024 – 30 August, 2024

*Uecker*: What are the post-meeting deadlines for?
*Seacord*: It is maybe a historical artifact at this point. We used to bundle together all the papers since the last mailing and it got sent out. We are not that critical anymore. We are less concerned about mailing deadlines and more concerned about hitting the deadline for having your paper review-able.
*Uecker*: Can we get rid of it?
*Seacord*: Are there any objections? That would mean just taking it off the agenda.
*Gustedt*: It would be good to have a deadline for the minutes.
*Seacord*: We can replace the post-meeting deadline with the minutes deadline.
*Bhakta*: I am happy that anyone volunteers to do the minutes and don't want to discourage them with an arbitrary deadline. I like *Uecker*'s suggestion in the chat: "The deadlines for the minutes can be the pre-deadline for the next meeting"
*Seacord*: I will make the change because it's become a bit silly.
*Bhakta*: *Myers* wants to know the end of the Minneapolis meeting to help with his travel times.
*Celeste*: The venue will be available until 6 p. m. but I was expecting that we'd stop at 12:30.
*Bhakta*: Thanks for keeping the room to the end of the day anyway.

## 5. Document Review

### 5.0 Ballman, alignof of an incomplete array type [N3053]

*Ballman* took an action item to champion it but then forgot. Odd thing if you think about it because the alignment of an array is the alignment of a pointer to a type, so you don't need to think about it. C should be able to handle incomplete array types but not incomplete types themselves.

This removes an existing restriction. There should be no chance of breaking existing C code. Multiple compilers already accept this. C++ already works this way.

*Myers*: I have an issue with wording of the example. Arrays in example aren't allowed because you can't have an array of incomplete type. You can't have it so can't apply alignof either.
*Ballman*: That's easy to resolve. We could vote on it in principle. or could vote on it a second time.
*Seacord*: That sounds complicated; could you not just do homework?
*Uecker*: I am fine with the paper.
*Gustedt*: I am also fine with the paper except that the example should not use "can" because that's not normative text.
*Jabot*: Are we just saying we want to remove the last example?
*Ballman*: We either remove or modify the last example and remove the word "can."

*Seacord*: Skip the straw poll.
*Bazley*: I am severely hampered from giving useful feedback.
*Seacord*: We will slot it in on Friday.
*Ballman*: The paper has waited since October 2018 so it can bake for a bit longer.
*Seacord*: I'd like to make progress if possible and move on to avoid taking a slot every meeting.
*Bhakta*: This is a procedural thing, it would be useful to mention what homework people have done at the beginning of a meeting.
*Jabot*: We did not request any change to the normative text, so I would be happy to poll right now with editorial change
*Bhakta*: We don't want to get into the habit that normative changes are the only changes that matter. If we are keeping the example then I would like to see the full paper to vote.
*Seacord*: Stick with the homework.

**5.1 Ballman, Allow zero length operations on null pointers [N3261]**

This document proposes to remove an instance of undefined behavior regarding operations you can do on null pointers. People might think that adding or subtracting zero from a pointer is okay, but it's not. memcpy(), memmove() and other standard functions are also undefined behavior if you pass in zero as a size. Everybody does the same thing, which is exactly what you'd expect. The goal is to remove that undefined behavior so that we can have safer C programs by blessing a lot of existing programs that already make these assumptions. This also unblocks optimization opportunities. We would currently would need to check for null pointers, to alert users. This would make those branches unnecessary. It's really about trying to remove a undefined behavior that doesn't seem to be holding its value. This undefined behavior is not really taken advantage of in practice. LLVM refuses to work with C libraries that don't do this. If the library doesn't handle null pointer added to 0 then I am happy not working with conforming libraries because of the lost optimization opportunities.

*Steenberg*: Are there any issues of knowing that something is null? Is the second zero in null plus zero also zero or is it null?
*Ballman*: In the first case, you can't add pointers, so second 0 is not a null. As *Gustedt* says, zero is not a pointer. It can be converted into one if you sneeze at it but it's not a pointer.
*Steenberg*: I am still concerned.
*Ballman*: If pointers are unrelated then that's still undefined behavior.
*Krause* brought up a problem with 16-bit pointers where some bits are stolen for non-null pointer address spaces. Address spaces are not part of standard C – only a technical detail that some implementations implement. Which sets of constituencies should pay the cost? Cost is currently on the vast majority of people's lives, or we can optimize for people who use platforms where null pointers are not zero. This makes a lot of sense to implement the usual semantics because they can still implement the address spaces but with a small cost. The security wins outweigh the cost.
*Krause*: Even if the AS isn't used by the user, it's still in the hardware. This claims that a lot of undefined behavior goes away, but the only place we have problems is small systems. If you subtract two null pointers that happen to be the same then you get actual zero. There is a difference if you do that. The paper says it's for consistency but it feels like there's not enough motivation about subtraction of null pointers. I can go along with adding zero to a pointer, but subtracting null pointers has an actual cost to change the implementation, so I would like to see a motivation where it is actually useful.
*Myers*: When comparing two null pointers, they already compare equal in the existing standard.
*Bhakta*: It's not only small systems but there are also big systems that have the same thing.

*Ballman*: For subtraction, if address spaces are disjoint, then you've already hit undefined behavior; if they do overlap, then you already have to have the extra instructions in place to handle it. Whether people are implementing the tech report is another matter. This may not be as big an issue as it seemed it was.
We want subtraction defined for self-consistency. We don't have a specific need. We are not leaving optimization opportunities on the table but *Popov* can correct me.
*Uecker*: Sanitizers and tools. Optimization is one use, but another is inserting checks at runtime. Now things become more complicated. Do you think it's a problem and does it have an impact?

*Ballman*: Tools that are looking for mistakes will have to change because the kind of mistakes you can make are different. I don't know whether that's necessarily a problem. Removal of any undefined behavior would have the same effect. This is already the existing behavior in C++ which has a cost but I not aware of it having any impact. Slow path so doesn't matter?
*Popov*: When we implemented the null pointer checks in Clang, we also implemented optimizations in LLVM to reduce overhead. There was still some overhead afterwards but I am not sure of the figures.

*Gustedt*: Pivoting back and forth about the subtraction of pointers and the importance of that. We may be invalidating existing implementations. I am not comfortable with that. Let's track this subtraction thing separately. With the others, I'm fine, at least I can see what that implies for library implementers to handle null pointers in certain situations. This seems quite innocent to me.

*Ballman*: This does not invalidate user code; only implementation practice, but it is special because it may have performance impact.

*Gustedt*: We allow it but I don't think the reason is good enough.
*Ballman*: I would challenge anyone in the room to ask a C user what you get when you subtract two null pointers? It's pretty weird that it's undefined behavior to subtract a null pointer value from itself.
*Gustedt*: Now you're doing the same error as before – null pointers can have different values.
*Ballman*: According to the tech report, when those pointers are in separate address spaces, that's already undefined behavior. It only affects overlapping address spaces.
*Myers*: I am concerned about wording of library functions. It is not clear that new wording about length of the array would cover the new use cases.

*Bazley*: Our default position is always to remove undefined behavior but I am concerned about the proposed changes to the library. (Much not heard by others.)

*Ballman* (interpreting): "Let's make sure we separate out the subtraction side of the changes."

*Bazley*: My main concern is null pointer analysis. Allowing null parameters to memcpy() et. al. could set a precedent for the whole library. It cannot be checked in the implementation just by checking the maybe-null pointer but would rely on the value of other parameters.

*Krause*: Address spaces are used when users need it for performance optimization.

*Ballman*: Is there anyone with over-my-dead-body objections, leaving aside the subtraction question?

*Jabot*: It's weird to keep undefined behavior around so that it's easier to detect undefined behavior. We should take the opportunity to increase safety where possible.

*Celeste*: I have no problem from the perspective of an analyzer implementer. Where the library functions would start admitting null, it would be for the same reason that it would be allowed outside the function, which is better than a set of discrete accesses that may or may not have meaning.

*Ballman*: The benefit of adopting this including the subtraction side of this is that it benefits those working in C and C++.

*Krause*: I also don't fully buy the consistency argument for subtraction. Sure N + 0 will become a null pointer if N is a null pointer, but I think typically N + 0 would be the same null pointer as N, which doesn't tell us anything about what should happen if we subtract two different null pointers.

*Ballman*: I don't imagine we'll make any distinction between C and C++ in the undefined behavior sanitizer. The sentiment seems to be to hear something along the lines, and we can split out the straw poll on subtraction.

*Uecker*: I think it is nice to be compatible with C++. (It would be even nicer if WG21 would talk to us when making such changes.)

**5.2 Uecker, A String Type for C [N3210]**

*Uecker*: A length-prefixed string would be nice for safety. I thought it would start a discussion and motivate others. My motivation was twofold:
Avoid index computing by hand because of security (often forget a one or whatever). The idea is to do that more safely but using safe API's in the standard library
At the same time I wanted to address the need to have a low-level type for memory management or at interface boundaries to other languages and systems. I wanted something simple and low-level. We can have a string type with a size and array member. It is called a Pascal string.
The idea is to specify this as a representation of the string so there is no implementation freedom left. This has advantages and disadvantages. At the same time it discourage people from messing with it by using an incomplete struct type in the header. People are then pushed towards using the API.
We want to have a type that works today so that it doesn't rely on future language editions.
We could also add a string-view type although not proposed. The open design allows extensions by us or by the users.
Functions allow strings to be created from UTF-8 or get back a C string. This makes it easy to pass it to an existing API. Other helper functions get the length or concatenate strings. There are helper functions to compare strings. Users can mainly use safe functions.
We Should focus on UTF-8 encoded strings. It doesn't make sense to have alternatives. We have conversion functions.

*Krause*: I disagree with what you said about UTF-8. People are using it anyway but are using strings of char. We should not put that requirement into the string data type. Instead we should have a type that replaces an array of char that is compatible with old interfaces. There are still use cases for other encodings in the embedded area. Sometimes we use other encodings if they know they will use only some characters.

*Uecker*: We avoid problems by fixing different systems to use UTF-8.

*Gustedt*: I'm much in favor of this proposal in one way or another, whether or not we use UTF-8. Exactly how I program strings using flexible array members. There are some inconsistencies in the proposal: sometimes "length", sometimes "size" of a string. I really like that as a simple proposal for doing things.

*Ballman*: I think it's valuable. It's an illusion to think that one size fits all. It may not be appropriate for all systems or languages. It's useful and simple. Being simple is a big advantage. I like that it provides an ABI too. There's prior art in an early stage. The compare function is under-specified.

*Douglas*: I quite like this approach compared to other approaches. What happens if you have a billion strings of one character long? This seems like overkill. A different form of encoding the size would get the size down. And throwing a size_t at it seems expensive in 64-bit.

*Uecker*: Simplicity is more important. You can roll your own type if you need to. Pascal started with 8 bytes. Go with size_t.

*Svoboda*: There are several other implementations of strings. Think of Annex K. C++ and QT, GTK have their own implementation. String types in Java, Python, Rust. There are lots of different ways of solving this problem. I'll agree that you can't make everybody happy. You not only have to prove that your approach is good and solid, but it is better than other approaches. I would expect to see discussion of other approaches and why you didn't take them. You clearly aren't trying to capture strings on the stack or char arrays on the stack. That's fine but be clear about it. It's hard to do well. Convince us that it belongs in the standard unlike every other type that has been proposed and has been built.

*Uecker*: It's the simplest thing you can do. It will not make people happy who want additional functionality.
You can cast anything to const and it just works, but I'm not sure that's what you meant.
If we get the byte array proposal then you could put it on the stack although that is not the best way to use it.

We can also have higher-level types such as that proposed by *Bazley*. That doesn't mean we cannot add something else as well.

*Svoboda*: The best thing is to say, "Feel free to add other solutions."

*Seacord*: Is there a way to make this work with static strings?

*Uecker*: Not at the moment. some compilers do but that's not standard-conforming.

*Cranmer*: How might this impact the ability to express ABI's? There's a useful distinction between a UTF-8 slice and a string type. Would not want to give more thought to encoding. Always UTF-8 – do not provide UTF-16 or UTF-32.

*Bazley*: Thanks to *Uecker* for throwing down the gauntlet. If anyone wants to know what I think and they have several hours to spare they could read the 70 page paper I accidentally wrote. I feel that *Uecker* solved the wrong problem because what matters is the interface, not the implementation which implementers should be able to tailor for the target platform. I want to see prior art for all serious proposals. Some prior art I reviewed changed my ideas instead of doing things like I always had. I think that people will always work with char arrays; new char8_t functions would join the large amount of stuff in the C standard that neither I nor anyone I know has any use for. Worried that functions which return pointers to new allocated blocks will lead to dangling pointers and leaks.

*Na*: I agree with *Bazley*. string_from makes the user free the pointers. This causes temporal safety problems, which might worsen C++ compatibility.

*Uecker*: Temporal safety needs a general solution in C. It would work for everything we allocate and free in C.

*Na*: It is more implicit that the string functions allocate extra memory.

*Łukasiewicz*: I like the idea of making something with strings. It's a pressing issue with C. A bit too early to vote something in. Proposal wanted to avoid using new functionality of C2y but there might be something in the new standard that could greatly improve the string implementation. For example fat pointers. Wait for new mechanisms to see if something should be useful.

*Celeste*: I prefer *Bazley*'s solution for the reason you gave which is that C is the language of manipulating data structures. If we bring in something at all, bring in something complete instead of a simple thing. More meaningful choice than giving users something lightweight.

*Uecker*: I would add *Bazley*'s proposal and try to work together. A lot of existing code would not be rewritten. Feeling is that something more lightweight would be more likely to be used.

*Straw Poll*: Would WG14 like to see something along the lines of N3210 in C2y? 10/4/10.
*Seacord*: Consensus but with a strong tail of people who aren't sure what's going on.

*Kaye*: I just wanted to note that teach-ability is important, and that the focus on UTF-8 is helpful for that. Leaving the encoding unspecified by using "char" invites problems. If necessary then we could take inspiration from C++'s u8string, u16string etc. – it's not pretty, but it makes those encodings explicit, which can help prevent errors.

**5.3 Uecker, Memory-Safety in C [N3211]**

*Uecker*: The idea is to provide a memory safe mode that is pragma-activated.
Two modes:
The static mode requires the compiler to reject everything that causes undefined behavior at compile time.
An incremental way is the only way to evolve it with the resources we have.

Dynamic mode builds on runtime checks that already exist in the form of sanitizers. This would change the semantics of operations that cause undefined behavior now to trap instead (e.g. on overflow). This can make dynamic mode memory safe but the compiler would be required to add checks similar to sanitizers.

*Bachmann*: I like the idea but it should be more comprehensive. I wouldn't like to see random annotations added to the language. This should be a coordinated effort, not adding a few annotations here and there. I like the incremental approach.

*Svoboda*: This is not actually going to add any features to the language itself.

*Uecker*: Not part of this effort but adding wide pointer would extend the possibilities.

*Svoboda*: People are saying, "You're never going to get memory safety in C and C++ so you should use Rust." I like it because it shows what you can do in C. My advice is to contact people in WG21 so they can add it to C++. I would like to see it move forward to spur a market for memory-safe C.

*Uecker*: Talking to C++ is a good idea.

*Celeste*: I like this. Can we use annotations? That could be more composable than working in the pragma space. Allow list is similar to things allowed in constexpr functions. It Seems like things that should be directly tied to each other.

*Uecker*: Good idea about constexpr. I'm not sure what the best approach is about pragma and attributes. Pragma can be statement-level. Attributes strike me as less flexible but I am open to ideas.

*Bhakta*: One of those things that would be great for a TS. To see how people implement it, if it works, instead of having in in the main C standard.

*Cranmer*: My first take is that it's a good idea for exploration. I worry when I look at the memory safety stuff how easily you can define it so that you can safely dereference a pointer?

*Uecker*: The general strategy is to add annotations to a function. It shifts the responsibility in the safe mode to the caller. This is the only way to achieve this because we cannot strengthen the semantics of the C language because of backwards compatibility. By defining the rules of the safe mode then if the caller is in the safe mode then we can say that it is defined by the rules of the language.

*Na*: In the C++ committee there was a similar paper for contract preconditions. Would it be a good starting point? Have you considered fine-grained opt-out annotations? e.g. functions might be mostly safe except for one line.

*Uecker*: The idea with the pragma's is that you can turn it off for one line. We could also consider other ways. I'm not sure what the status of contracts is. If C gets contracts then it could play a role extending this functional. Contracts alone would not ensure memory safety. Contracts make sense but hasn't been discussed in the C committee.

*Opinion Poll*: Would WG14 like to see something like N3211? 16-2-6 Pretty strong consensus.

### 5.4 Krause, Restrict atomic_flag creation [N3246]

*Celeste*: Useful for guarantees provided, and if the guarantees cannot be provided, there is a problem. Given that this atomic-flag guarantee is most important at the low end, we need to change.
*Bhakta*: I agree with Alex, but I'd like to get the sense of the room?

*Opinion Poll*: Do we want to see something along the lines of N3246 in C2y? 15-0-8

**5.5 Ballman, Generic selection expression with a type operand [N3260]**

*Celeste*: By opening up to matching complete type, you open up to void. This took me less time to add to the compiler than it would take a user to add it to their code.
*Łukasiewicz*: Matching against arrays would be helpful, does this work?
*Ballman*: Yes, you can match against specific array sizes.

*Svoboda*: Does lvalue-rvalue conversion happen with sizeof / offsetof, like generic does?
*Ballman*: No, lvalue-rvalue conversion does not happen with sizeof
*Svoboda*: So that difference will persist through your proposal, then?
*Ballman*: Yes

*Gustedt*: Should we add recommended practice for compilers? Is this being used in practice?
*Ballman*: I like the idea; we should have more recommended practices in general. It's a quality of implementation argument though. I'm not sure that my user is using it. I've been using it a lot, but then I test compilers.

*Bhakta*: Good idea, but it changes existing code. Have you done code searches for dependencies on this change?
*Ballman*: How does this change existing code? I have not done a code search. This went out in Clang 17.
*Bhakta*: Oh, you're right. it doesn't.

*Celeste*: Regarding recommended practice: MISRA 23.4 rule warns on loss of qualifiers (atomics, lvalue-conversion). I'd prefer moving this to C, not MISRA.

*Bazley*: This won't appease my colleagues who want polymorphism. But it is useful for combining existing features.
*Ballman*: Yes, this doesn't solve polymorphism.

*Myers*: Regarding warning for unreachable code, most recommendations are for macro expansions.
*Ballman*: Regarding recommended practice, we should exclude recommendations for macros. Recommended practice should be separate from this paper.

*Bazley*: I am confused about the example. What is the purpose of these expressions?
*Ballman*: We can't have "generic" by itself at the file level. So I put them inside a function. I would like to avoid an example wrapped inside a macro; they add extra abstraction.
*Bhakta*: In our examples, we leave out functions, and just have "i" defined.
*Gustedt*: Agreed.
*Svoboda*: I disagree, the function indicates "i" is initialized with an unknown value

*Straw Poll*: Does WG14 want to adopt the proposed wording from N3260 into C2y? 19-0-4 Passes

**5.8 Łukasiewicz, Seacord, The C Standard Charter [N3255, N3223]**

*Bhakta*: What does "having a charter" mean? There's no real enforcement.
*Seacord*: Lots of painful changes, and this is the least important thing we do. It's almost more of a PR piece.
*Gustedt*: Even if just for PR, it is important to have a charter.

*Bazley*: The process of making the charter is the most important part. I often read the charter/manifesto as my local SCRUM master.
*Ballman*: I am uncomfortable with "Do not leave features in an underdeveloped state." This feels like a slippery slope to me. Can we agree that the goal of that point is not to allow feature creep?
*Seacord*: Yes, we agree this isn't binding. The charter is mostly a cudgel that people use to argue for their paper, we have never followed its principles strictly.

*Bazley*: The WG14 website isn't very friendly. It looks like an accretion of stuff over history, which is not a good format for a charter. A new up-to-date charter is therefore good.
*Seacord*: Are you volunteering for the infrastructure group to update the website?
*Bazley*: The current charter is not good.

*Kaye*: "Don't let the perfect be the enemy of the good."
*Seacord*: This committee is afraid of big changes. So people sneak little changes, and then bully the rest of a big change through.

*Pygott*: The charter is useful as a public statement of our aims and objectives. We must consider political appearance of the charter. Your government is forbidding things that are not "type-safe."
*Seacord*: That is the Five Eyes nations, not just the US government.
*Celeste*: A lot of C design decision were made based on "the perfect being enemy of the good." See the Unix design principle "Worse is better."

*Anley*: There is language about safety. But I didn't see anything about security…should we add that?
*Seacord*: We do have "all known software security issues."
*Łukasiewicz*: Functional safety is also a security issue.
*Anley*: Perhaps we should emphasize security more, e.g. "safe and secure programming."
*Seacord*: Are there any objections to "enable safe [and secure] programming"?
*Bazley*: It would be clearer to have "enable secure programming."

*Seacord*: Do we link to the charter from the website?
*Ballman*: Yes

*Straw Poll*: Do we want to adopt N3255 as the "official" C standard charter with "Enable safe programming" changed to "Enable secure programming"? 17-0-6 Passes

*Seacord*: We'll need a new document with the changes.

*Kaye*: Regarding migrating to newer language editions, for the statement: "Developers should be able to mix and match code from different editions" Does that mean every single edition of ISO C? Can they skip intermediate editions? The text is ambiguous.
*Bhakta*: This is a charter-general thing. We don't want to give specific directions about migrations. I would prefer not to change anything.

**5.11 Thomas, Proposal for C2y – Round-trip rounding [N3232]**

*Seacord*: This is a clarification, not a change to the standard.

*Straw Poll*: Does WG14 want to adopt the proposed wording from N3232 into C2y? 21-0-2 Passes

**5.13 Thomas, Proposal for C2y – Recommendation for printf rounding [N3233]**

*Seacord*: Is the use of "should" equivalent to "recommended practice"?
*Bhakta*: Yes, but it is restricted to where it can be used. Ask Meneide or Wiedijk.

*Bazley*: Is this easy to write in prose, but with implementation constraints? I am concerned about corner cases.
*Bhakta*: I know Tydeman has a test suite that people are using. In practice, it is not an issue. There are some degenerate cases where you need a huge number of digits. Tydeman may have a more authoritative answer.

*Douglas*: If you change how floating-point numbers get printed, won't this break a lot of test suites?
*Bhakta*: Yes, this is recommended practice. It gets people more accurate results (if there is a change).
*Tydeman*: I don't recall this being an issue, but I am not sure I tested this corner case.

*Seacord*: This is recommended practice. It doesn't do any harm, or great benefit. I have no problem with including it.

*Straw Poll*: Does WG14 want to adopt the proposed wording from N3233 into C2y? 20-0-2 Passes

**5.18 Thomas, Problematic use of "correctly rounded" [N3242]**

*Straw Poll*: Does WG14 want to adopt the proposed wording from N3242 into C2y? 18-0-5 Passes

**5.14 Gustedt, The semantics of the restrict qualifier [N3234]**

*Bazley*: Revising this text will be a lot of work. Is it worth it because beneficiaries are implementers. What do implementers think? I could use some examples to understand this.
*Seacord*: In the standard examples are non-normative.

*Cranmer*: The wording must describe when pointers point to the same object. This is not the same as a data dependency relationship. You don't want the standard's definition of alias analysis to be too restrictive.
*Gustedt*: The intent was to cover same ground as original text, neither more nor less restrictive, using more modern normative text. The old text was also difficult to read, and also nonsensical.

*Ballman*: Yes, the current wording is a problem for implementers, improving wording is appreciated. I am concerned about basing this off of the current memory model, we know that the memory model wording is relaxed, that's why we have TS 6010. Many C++ compilers also support "restrict." Overall, I want more clarity on the restrict qualifier.
*Gustedt*: This was the closest I've found for the carries-dependency from the original text.

*Bhakta*: When we implemented restrict, we found customers had more issues with restrict than we did. Their understanding was often incorrect. In practice, it did improve performance in customer code. As long as semantics stay the same, we are good. I think user comprehensibility outranks implementer comprehensibility.

*Myers*: I provided on the reflector an example of multiple pointers. I'd like to see how the wording applies to such an example, of multiple levels of restricted pointers.

*Wiedijk*: One student wrote his master's thesis on "restrict" with formal semantics. I could send it to people here. Who would be interested?
*Gustedt*: I'd be interested. We can work together on the paper.

**5.11 Gustedt, Some constants are literally literals v2 [N3239]**

*Ballman*: Do integer literals include the suffix?
*Gustedt*: Yes

*Myers*: There are typos in the sub-clause numbers.
*Gustedt*: Possibly so. This can still be reviewed.

Opinion Poll: Does WG14 want to adopt the proposed wording along the lines (subject to review) from N3239 into C2y? 15-0-10
*Seacord*: Can anyone explain why they chose to abstain?
Johnson: Yes, I'm looking for a good definition between "literal" and "constant."

**5.12 Gustedt, Remove imaginary types [N3263]**

*Myers*: N3263 doesn't include all items in previous document (N3240).

*Bhakta*: CFP disagrees with the removal of Imaginary types. They are useful for efficiency and accuracy. Imaginary types are implemented by HP.
*Gustedt*: Efficiency is good but only if used and implemented. Regarding the HP implementation, no one could give me a link for finding and testing their compiler.
*Bhakta*: Here is the URL for the HP compiler: https://www.hpe.com/psnow/doc/4aa3-9071enw.pdf?jumpid=in_pdfviewer-psnow

*Tydeman*: By changing "I" to be Complex you are changing semantic of IEEE-754.
*Gustedt*: We only change this for platforms that had "Imaginary" before.

*Myers*: If we keep imaginary types, we should separate things out from Annex G.
*Ballman*: I support this, it makes the language simpler. Clang had one user who wanted support for imaginary in 2010, because it was in the standard, even though they had no use for it.
*Tydeman*: Some of us in CFP noticed that some items in Annex G are separate from IEEE-754, and could be moved to the main part of the standard.
*Gustedt*: I would support such a movement.
*Tydeman*: Wakely posted on the reflector that the EDG front-end supports imaginary types.
*Celeste*: We (MISRA) never received bug reports about missing Imaginary.
*Johnson*: The Cray compiler (being Fortran/C/C++) supports Imaginary. No numbers on users. We could probably remove this.
*Gustedt*: Does this support C23?
*Johnson*: The old HP_UX compiler would accept bug reports (but it is not actively developed). Cray has a large lively user base and compiler development team.
*Bhakta*: I am volunteering CFP to work on migrating parts of Annex G into the main standard.
*Gustedt*: We have a question of character i mapping to complex i. I could do this as homework.

*Opinion Poll*: Would we like to remove imaginary types from C2y along the lines of n3263? 14-2-7

**5.13 Gustedt, Introduce complex literals [N3241]**

*Krause*: Why add so many suffixes (for complex types)? Why have both "i","I" and "j","J"?
*Gustedt*: This is a compromise. In engineering, they use "j" or "J."

*Celeste*: I am not convinced we should list every permutation of integer literal suffixes. For example, mixing capitalization is hard to read.
*Gustedt*: I understand, but I did not want to invent a new grammatical term, so this was the easiest way to describe the literal suffixes.
*Bhakta*: Some comments from CFP: There are concerns with the imaginary suffix, a suffix constant behaves like a complex value, rather than a pure imaginary value. They are a complex value with real-value 0. There are also issues with special values (e.g. -0, + 1I) does not have a -0 real part. I * NAN has a real part.
*Gustedt*: How is that necessary? I don't understand the problem here.
*Bazley*: I don't see how we can have so much redundancy on the language. We should do what C++ does.
*Gustedt*: C++ is not compatible regarding complex numbers. We have "F" and "L" here.
*Cranmer*: I found a case with lots of infinities and zeroes. INFINITY * I is a complex multiplication (not imaginary).
*Gustedt*: Would it help to see an example in the standard to explain that?
*Cranmer*: I recommend that a user always uses a complex constructor than "i","I" suffix.

*Cranmer*: If you do a+b*I is not construction of complex number.
*Myers*: This version means implementations need to check for versions of C2y. Seems useless to me.
*Ballman*: By disallowing it, we have to issue pedantic diagnostics. Clang and GCC support both "i","I" and "j","J" as suffixes.

*Straw Poll*: Would WG14 like to adopt the wording in section 4.1 of N3241 in to C2y? 3-9-13 non-consensus

*Opinion Poll*: Would WG14 want to also have "j","J" in addition to "i","I" in suffixes in N3241? 8-8-9

*Ballman*: It is interesting how implementers voted frequently "yes", but others voted "no." Is saying "no" based on language purity or practicality?

*Opinion Poll*: Would WG14 want to allow only "i", "il", and "if" (without capitalization) as suffixes in N3241? 4-10-10

*Opinion Poll*: Would WG14 want to allow only "i", "il", and "if" (and with capitalization) as suffixes in N3241? 7-8-9

### 5.14 Uecker, Polymorphic Types [N3212]

*Celeste*: I support this feature. The atomic function case is enough of a use case by itself, so is tgmath. It is not good for standard API's to require compiler magic. I don't see the value instantiation of variably-typed objects at compile time. It's equivalent to something like aligned_alloc(). Does not add type safety or type checking. We need a generalized solution for dependent types.
*Łukasiewicz*: I also support this paper. It integrates with current semantics of C, it gives opportunity for interoperability, which C++ templates don't support. I request not making struct type dependent on the library.
*Gustedt*: The choice of keywords is no good. Something called "type" which is type ID is not a good idea, likewise something called "var" which is a type. I posted to the reflector flaws in the examples. My concern is: I don't see how I could use it, so I could forward arguments to the function. There are no operators defined that make the type ID really useful. I would need something like _Generic in my function. In qsort() example, you have type-generic functions; can it benefit?
*Uecker*: I don't care about keywords. You could allow copying if you know the size.
*Seacord*: I recommend that you let the questioner comment; you're not going to change opinions in the meeting.
*Myers*: There is no specification wording yet. Definitely implementation experience will help. Needs ABI's too, although that is out of WG14 scope.
*Anley*: It seems helpful on security grounds. Void-based polymorphism is a major source of vulnerabilities. We get hundreds of C codebases annually, and see issues related to a lack of this feature in the language.
*Johnson*: Was this modeled after Fortran polymorphism?
*Uecker*: No, it was more based on Zig.
*Ballman*: I recommend that if we move forward, we should form a study group and publish a TS. One concern: the design has some novel bits. It mentions about constructing types at compile time, but limit to circumstances available at compile time is strange. I don't know how you would support this code example:

```
// TU1.h
void func(_Type Ty, int V1, int V2);

// TU1.c
void func(_Type Ty, int V1, int V2) {
  _Var(Ty) Value = { V1, V2 };
}

// TU2.c
#include "TU1.h"

struct S {
  float a, b;
};

int main() {
  func(_Typeof(struct S), 1, 2);
}
```

For Clang, we get push-back on RTTI, and have to provide a "–no-RTTI" option, sometimes for low overhead, and sometimes for security. If we can avoid RTTI that would received better than following C++'s RTTI.

*Ballman*: If others think a study group is useful, we could vote on that now.

**5.15 Seacord, Accessing byte arrays, v4 [N3254]**

*Celeste*: Regarding Copying types: The Distinction between memcpy() to new location, vs. memcpy() to temporary location, changing, and then memcpy() back to original location. The latter case is not necessarily broken for reasons the former case is.
*McKinney*: We use this a lot, even though it is undefined behavior. We normally do this in-place.
*Steenberg*: This is very difficult and unintuitive. It may break backwards compatibility. A gulf exists between what the standards says and what people do. I think nobody understands them and writes to them. Compilers therefore do not optimize to the words as stated. So breaking compatibility may not matter in reality. They allow you to change effective type. This paper makes behavior more compliant, sort of by accident, because people don't know these rules.
*Seacord*: MISRA has a "no undefined behavior" rule. So at Toyota we can't convert bytes to an array, but this paper will let us do that.
*Bachmann*: I am in favor. I don't share *Steenberg*'s concern. I'm not sure my interpretation is correct. Believe that union typing only works when you copy data into a union. This text is more liberal. Not sure to what extent previous compilers already did this, including where a byte array is in a struct. I would like to see a feature-test macro before C2y is published.
*Seacord*: I couldn't find any platform where the undefined behavior was taken advantage of.
*Bazley*: I am bemused by this paper and by a lack of prior art. Our changes should be driven by use cases. I write serialization code, operating on each individual char to handle endianness. So I am concerned that this makes the task of writing correct C harder.
*Myers*: I commented on *Celeste*'s comments. Copying elsewhere and then back can cause problems. I don't think this paper has those problems anymore.
*Steenberg*: Despite my concerns, I'm not against the paper.
*Uecker*:GCC would treat structs with byte arrays for optimization. C++ also has this functionality.

*Straw Poll*: Does WG14 want to adopt the proposed wording from N3254 into C2y? 17-0-8 consensus

**5.17 Bazley, Standardize strnlen and wcsnlen [N3252]**

*Ballman*: We have precedent for bringing Annex K functions into normative C, see memset_s(). Annex K is a really useful source for us.
*Stoughton*: I am in favor of adopting the POSIX versions of these. The wording here is subtly different. Are this wording and POSIX's wording in alignment.
*Bazley*: I did change the wording.
*Myers*: Note that if we accept them, the next proposal for zero-length operations should include them.
*Seacord*: We had a proposal bring functions from POSIX into C. We accepted some but rejected most, and that includes strnlen(). That paper did not propose wcsnlen(). The discussion at the time was that strnlen() had some well-known security flaws, comparable to strncp()y/strncat().
*Bazley*: There is nothing inherently wrong with functions designed to run on fixed-length strings. Impossible to implement strndup() without strnlen().
*Seacord*: The only function that is inherently insecure is gets(), which has been deprecated. All other functions have misuse cases, as opposed to being insecure.
*Douglas*: My problem with strnlen(): These functions are easy to misuse. I would prefer using assertions or abort() if a string function is too long. There should be a way for POSIX compatibility without sacrificing security.
*Svoboda*: I missed that POSIX-functions-to-C paper. strncpy() and strncat() are vulnerable because they can turn a null-terminated byte string into a non-null-terminated byte string. strnlen() can't.

*Meneide*: I got email asking if we standardized strndup() without a wide version, so please keep the wide version (wcsnlen()).

*Bhakta*: Given the difficulties with agenda and the paper. I haven't been able to review, so I don't want it in C2y yet.
*Ballman*: Can we adopt it in principle, assuming no objections before next meeting?

*Straw Poll*: Does WG14 want to adopt the proposed wording from N3252 into C2y in principle? 19-0-6 consensus

### 5.16 Bazley, strb_t: A standard string buffer type [N3250]

*Anley*: I strongly support this from a security point of view. We get push-back about using C in our environment.
*Svoboda*: I like this proposal, mainly because it solves just a subset of strings, which is much easier than trying to solve the entire problem of implementing strings. Mutable strings are demonstrated in Java and Rust. Error-handling will be a problem (bad error-handling seems to be what doomed Annex K, based on Martin Sebor's paper). Your "local error handler" is one of the least bad strategies. The proposal still needs normative text.
*Ballman*: We need something in this space. I am concerned about statefulness. One other strategy to consider is twines, you keep pieces of a buffer. This is not possible with statefulness.
*Myers*: We definitely want some managed string type. Is there implementation experience? Can we get this into a C RFI?
*Celeste*: C RFI is stalled, because of the complexity of managing arbitrary users and contributions. It will get set up at some point. I'd like a stateful one, a PL/1 one, hopefully by next meeting.
*Bazley*: If you don't like statefulness, how would you solve the same problem? Appending to the end of a string is stateful.
*Meneide*: I worked with someone with a C codebase for an MMORPG; they had C# background. I had to manage way too many string-related bugs from this person.
*Bazley*: The behavior that STRB produces is very bounded.

### 5.17 Svoboda, Writing to multibyte character files [N3064]

*Myers*: From my reflector comments, write() refers to POSIX, and should instead be to fputc() and fputwc().

*Svoboda*: Yes. I would say fputs(), but that is defined in terms of fputc(), so OK.

*Straw Poll*: Would WG14 like to adopt recommendation 1 and 2b with “write()” replaced by “fputc() and fputwc()” into C2y? 17-1-6 passes N3064

*Pygott*: I am against this because I haven’t seen this paper before and have missed the last few undefined behavior meetings. But does this really remove undefined behavior? If you no longer have a valid set of multibyte characters, you still have something to worry about, even if it does not formally invoke undefined behavior.

### 5.18 Douglas, fopen "p" and bring fopen’s mode closer to POSIX 202x [N3247]

*Ballman*: I see the two behaviors being that duplicated chars are honored vs. not honored.
*Svoboda*: I thought the order of mode chars was arbitrary? I prefer it not be arbitrary, but doesn't this break code?
*Douglas*: Mirroring what POSIX does, the first letter is the only one with constraints. We actually give more flexibility.
If WG14 changes the wording would POSIX match it?
*Stoughton*: POSIX already published their 2024 standard, so it's too late.
*Bhakta*: Is MS the only one that implemented "b"? Our Zen-OS does too.

*Straw Poll*: Does WG14 want to adopt the wording from N3247 section 1.1 into C2y? 17-1-3 passes

*Svoboda*: How does this compare to Annex K's "u" mode fopen("u")? How does it compare to fopen("x")'s security?
*Stoughton*: How does this compare to tmpfile()?
*Douglas*: tmpfile() gets a name in a temp dir.
*Gustedt*: We need text to be consistent with the function in Annex K. It is still in C23.
*Douglas*: OK

*Straw Poll*: Does WG14 want to adopt the wording from N3247 section 1.2 and section 1.3 into C2y? 19-0-5 passes
*Action Item*: *Douglas*: Write a similar proposal to [N3247 using wording from Annex K.

**5.19 Steenberg, A Memory model with Synchronization based type aliasing V1 [N3243]**

*Gustedt*: I find this idea interesting. Do you also apply sync barriers to function parameter arguments?
*Steenberg*: Probably, but I'm not sure. Assigning a pointer value is a barrier, so this should solve itself.
*Gustedt*: The "fast math" option allows people to choose their level of performance. Maybe you can do something similar?
*Myers*: The GNU may-alias attribute is a possibility, it is not OK to ignore. An operator that address a non-addressable lvalue. It's not just effective type rules involved here. The result of converting pointers between types are not always defined.
*Douglas*: There are a lot of claims that strict aliasing is not popular. Agreed for legacy code, but for new code? (including all C++ code I've seen). All new code I've seen since 2011 is compatible with strict-aliasing. Just one major compiler does not implement strict aliasing optimization. I wonder if strict aliasing is as bad as the "vocal minority" claim?
*Steenberg*: The vast majority of C programmers do not know what "effective type" is. People don't know all things that are undefined behavior, because this all "just works." There is a lot of code out there that is technically undefined behavior but doesn't break.
*Bazley*: I also never had an issue with strict aliasing. But last week I discovered that the Marty-GPU compiler defaults to no-strict-aliasing. Unless this appeases the Linux kernel community, it is DOA; they are the main ones clamoring about strict aliasing.
*McKenney*: Never say to the Linux kernel community: 1. strict-aliasing is for people who don't understand typing, and 2. optimization is about safety.
*Bhakta*: It depends on your customer base. We have big banks that do want high performance and do understand strict aliasing. 99% of aliasing bugs are because people don't understand strict aliasing, but people are willing to learn it. Maybe general codebases don't care as much. We have multiple modes in our compilers because of this dichotomy.

**Gustedt, Inline functions accessing identifiers declared with constexpr [N3253]**

*Bhakta*: I agree with your question 2. For the 3rd bullet, footnote 2, what is the feature supposed to mean? It is not clear.
*Gustedt*: "Object" or "function"
*Celeste*: Specifically about things that have an address. As currently written, this will prevent clang constexpr functions from an inline function. Currently correct as written.
*Gustedt*: If we want constexpr functions we have to change the function call anyhow.
*Myers*: Regarding wording, "feature" used in the standard means for features in the language, not objects or functions. "Evaluation" was intended to make it unambiguous exactly what counts. This use of "evaluation" is far from clear exactly what does and does not count. It is Not necessary to introduce this new undefined behavior.
*Gustedt*: I don't understand how other text is precise. Evaluation of identification is relatively clear. This text for me is much clearer than whatever we had before. We can replace "feature" with "object" or "function."
*Myers*: See 6.9.1p3, which contains a precise definition. More precise than an evaluation. I'd like wording as precise as 6.9.1.

*Gustedt*: I think "reference of an identifier" is confusing. We may disagree regarding clarity.
*Celeste*: Adding undefined behavior is undesirable. Can the added undefined behavior become a constraint?
*Gustedt*: I don't think so.

*Straw Poll*: Does WG14 want to adopt the constraints for inline definitions as proposed in n3253 with "feature" replaced by "object or function" for C2y? 2-4-20 no consensus, people are confused
*Svoboda*: I abstained via my insufficient understanding of the problem space
*Celeste*: The proposal is better than the current wording but not yet good enough for the standard.
*Bazley*: Concerned that the experts don't agree. I'd rather experts agree before we vote it in.

**Uecker, Slay Some Earthly Demons I [N3244]**

*Bhakta*: I have comments for individual items.
*Action Item*: *Seacord*: Write a paper to clarify implementation-defined vs. unspecified behavior; do they require the standard to document choices?

*Straw Poll*: Does WG14 want to adopt the proposed change for 6.3.2.1 p3 (Annex J, item 21) but not necessarily the subsequent additional change from N3244 into C2y? 19-0-4 consensus

- (21) Additional change: Clarify semantics of “address taken”

  *Svoboda*: I will encourage you (along with everyone else) to submit a document like this to the UBSG. We would be happy to review it. This saves WG14 some effort, and you get faster turnaround feedback. You just said that an uninitialized unused auto var is undefined behavior in normative text. (because you took out the "register" keyword)

  *Bazley*: "Had its address taken" seems redundant
  *Uecker*: Could mean object of the & operator, or some other way.
  *Gustedt*: Adding precision is good.

  *Straw Poll*: Does WG14 want to adopt the proposed change for Annex J, item 21, "additional change" from N3244 into C2y? 10-2-10 Consensus

**Ballman, Support ++ and - on complex values [N3259]**

*Myers:* You need to be clear that adding/subtracting 1 is a real type (not imaginary)
*Ballman*: That surgery is wider than this paper. That should be covered by existing wording. Do you want a future version of this paper that addresses your concerns?
*Myers*: This paper leaves the semantics unclear.

*Straw Poll*: Does WG14 want to adopt the proposed wording from N3259 into C2y? 19-0-4 consensus
*Action Item*: *Ballman*: Add more clear wording about semantics to address *Myers*'s comments to N3259.

**Seacord, Usability of a byte-wise copy of va_list [N3262]**

*Gustedt*: Not sure if this is a good choice.

*Straw Poll*: Does WG14 want to adopt the proposed wording from N3262 into C2y? 16-0-7 consensus

**Ballman, alignof of an incomplete array type (updates N3273)**

*Straw Poll*: Does WG14 want to adopt the proposed wording from N3273 into C2y? 18-0-5 consensus

*Myers*: I think this is a new feature, not a defect
*Ballman*: This makes migrating between C versions easier.
*Straw Poll*: Does WG14 want to add N3273 to the Extensions to Obsolete Versions of C list? 12-1-9 consensus

**Gustedt, Remove imaginary types v3 [N3274]**

*Bhakta*: I am speaking for *Johnson*. Implementation is there with HP, they use EDG front-end, which supports Imaginary, and uses that. The version of the compiler that supports Imaginary is not being actively developed.
*Bachmann*: Outside compilers, i is used for purely imaginary type, I think people should use the complex macro, normally used for purely imaginary parts. So I am opposed.
*Celeste*: This is probably too radical to propose: "I" being a macro, most people find it not useful but annoying, so we should take it out.
*Gustedt*: Agreed.
*Tydeman*: Based on results of my test suite. HP compiler is the only one that conforms to IEEE-754.

*Straw Poll*: Does WG14 want to remove optional support for imaginary types as proposed in N3274 from C2y? 15-2-6 consensus
*Straw Poll*: Does WG14 want to remove the specification of __STDC_IEC_559_COMPLEX__ as proposed in N3274 from C2y? 10-4-9 consensus
*Seacord*: Why did people abstain?
*Bhakta*: I abstained because it doesn't matter to me.
*Meneide*: It is not something that matters all that much.
*Straw Poll*: Does WG14 want to apply the editorial changes as proposed in N3274 for C2y? 18-1-3 consensus
*Bhakta*: Why did *Johnson* say "no" to the editorial change?
*Johnson*: I can't just reject the entire paper.

*Opinion Poll*: Should WG14 make the changes to Annex G to make it standalone optional support for IEEE complex conformance? 14-1-7 direction to proceed

*Opinion poll*: Should WG14 make a purely optional annex for _Imaginary type support alone? 8-7-7 no direction

**Uecker, Slay Some Earthly Demons I [N3244]**

- (40)

  *Bhakta*: For this extension, NaN's are define with payload being integer value, but not an integer. This wording would make that invalid.
  *Tydeman*: No, that's a floating type.
  *Bachmann*: This would make it invalid to convert with union types (n3254).
  *Uecker*: This is not a pointer to conversion to a union.
  *Bachmann*: OK I misunderstood.
  *Myers*: You should allow converting to void.
  *Uecker*: OK good point. Withdrawn.

- (56)

  *Ballman*: Regarding the generic changes, allowing it to take a type operand. It can use (incomplete) tagged types. This means you cannot have local tagged types being used.
  *Uecker*: This is about objects with incomplete type and no linkage.
  *Ballman*: The type is fine, it is just an object declaration. thanks.

  *Straw Poll*: Does WG14 want to adopt the proposed change for Annex J, item 56 from N3244 into C2y? 21-0-1 consensus

- (57)

  *Celeste*: The GCC behavior does have a meaning, but not considered compelling evidence that this feature is useful outside test suites.

  *Straw Poll*: Does WG14 want to adopt the proposed change for Annex J, item 57 option 2 from N3244 into C2y? 11-6-6
  *Seacord*: Let's try the other option poll.
  *Straw Poll*: Does WG14 want to adopt the proposed change for Annex J, item 57 option 1 from N3244 into C2y? 14-2-7 consensus to adopt option 1

- (60)

  *Opinion Poll*: Request that the editor remove item 60 from Annex J.2 in C2y. 15-0-6 direction to the editor

- (63)

  *Gustedt*: Aren't there any qualifiers for functions that are non-standard?
  *Uecker*: Good question.
  *Bhakta*: Yes, we do have non-standard qualifiers for functions, say to change 32-bit to 64-bit or vice versa,. We don't claim standards conformance for those.
  *Celeste*: I've seen other function qualifiers.
  *Ballman*: Has anyone checked extension language like CUDA for offloading?

  *Straw Poll*: Does WG14 want to adopt the proposed change for Annex J, item 63 from N3244 into C2y? 8-3-10 not consensus
  *Seacord*: Why did people abstain? Some are not sure about CUDA.
  *Krause*: There are bank functions; other compilers have multiple qualifiers for functions.
  *Uecker*: If there is a good use case, we should make it implementation-defined.
  *Krause*: implementation-defined would be good.

- (67)

  *Douglas*: I remember having to inline one external function declaration. So I like this.
  *Uecker*: GCC would warn about this.

  *Straw Poll*: Does WG14 want to adopt the proposed change for Annex J, item 67 from N3244 into C2y? 17-1-4 consensus

- (69)

  *Gustedt*: I like the alternative more.
  *Bhakta*: I also prefer the alternative semantics. But in isolation, you could have looser alignment, indicating a programmer error.
  *Celeste*: We added MISRA rules which are slightly stricter. So moving to constraint follows the same direction.

  *Straw Poll*: Does WG14 want to adopt the proposed change for Annex J, item 69 using the Alternative Wording on Semantic Section from N3244 into C2y? 17-0-6 consensus

  *Seacord*: That's all for now, *Uecker* please submit the remaining bits in separate document.

## 6. Clarification Requests

The previous queue of clarification requests has been processed.

## 7. Other Business

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

**N2658 2022/01/03 Bachmann, Make pointer type casting useful without negatively impacting performance - updates n2484**

**N2995 2022/06/22 Meneide, __supports_literal needs serious design tweaking and there are likely better, less overt ways to solve the same problem**

**N3095 2023/01/30 Meneide, Restartable Functions for Efficient Character Conversion, r11 Another update to this paper must be made**

**N3160 2023/08/20 Grüninger, Add min, max for integers to C**

**N3104 2023/02/07 Meneide, More Modern Bit Utilities, r2: Another update to this paper must be made**

**N3155 2023/10/14 Svoboda, Examples of Undefined Behavior**

**N3121 2023/04/23 Uecker, Forward Declaration of Parameters v2 (Updates N2780)**

## 8. Recommendations and Decisions reached

### 8.1 Review of Decisions Reached

*Straw Poll*: Does WG14 want to adopt the proposed wording from N3260 into C2y? 19-0-4 Passes
*Straw Poll*: Do we want to adopt N3255 as the "official" C standard charter with "Enable safe programming" changed to "Enable secure programming"? 17-0-6 Passes
*Straw Poll*: Does WG14 want to adopt the proposed wording from N3232 into C2y? 21-0-2 Passes
*Straw Poll*: Does WG14 want to adopt the proposed wording from N3233 into C2y? 20-0-2 Passes
*Straw Poll*: Does WG14 want to adopt the proposed wording from N3242 into C2y? 18-0-5 Passes
*Straw Poll*: Would WG14 like to adopt the wording in section 4.1 of N3241 in to C2y? 3-9-13 non-consensus
*Straw Poll*: Does WG14 want to adopt the proposed wording from N3254 into C2y? 17-0-8 consensus
*Straw Poll*: Does WG14 want to adopt the proposed wording from N3252 into C2y in principle? 19-0-6 consensus
*Straw Poll*: Would WG14 like to adopt recommendation 1 and 2b with "write()" replaced by "fputc() and fputwc()" into C2y? 17-1-6 passes N3064
*Straw Poll*: Does WG14 want to adopt the wording from N3247 section 1.1 into C2y? 17-1-3 passes
*Straw Poll*: Does WG14 want to adopt the wording from N3247 section 1.2 and section 1.3 into C2y? 19-0-5 passes
*Straw Poll*: Does WG14 want to adopt the constraints for inline definitions as proposed in n3253 with "feature" replaced by "object or function" for C2y? 2-4-20 no consensus, people are confused
*Straw Poll*: Does WG14 want to adopt the proposed change for 6.3.2.1 p3 (Annex J, item 21) but not necessarily the subsequent additional change from N3244 into C2y? 19-0-4 consensus
*Straw Poll*: Does WG14 want to adopt the proposed change for Annex J, item 21, "additional change" from N3244 into C2y? 10-2-10 Consensus
*Straw Poll*: Does WG14 want to adopt the proposed wording from N3259 into C2y? 19-0-4 consensus
*Straw Poll*: Does WG14 want to adopt the proposed wording from N3262 into C2y? 16-0-7 consensus
*Straw Poll*: Does WG14 want to adopt the proposed wording from N3273 into C2y? 18-0-5 consensus
*Straw Poll*: Does WG14 want to add N3273 to the Extensions to Obsolete Versions of C list? 12-1-9 consensus
*Straw Poll*: Does WG14 want to remove optional support for imaginary types as proposed in N3274 from C2y? 15-2-6 consensus

*Straw Poll*: Does WG14 want to remove the specification of __STDC_IEC_559_COMPLEX__ as proposed in N3274 from C2y? 10-4-9 consensus
*Straw Poll*: Does WG14 want to apply the editorial changes as proposed in N3274 for C2y? 18-1-3 consensus
*Straw Poll*: Does WG14 want to adopt the proposed change for Annex J, item 56 from N3244 into C2y? 21-0-1 consensus
*Straw Poll*: Does WG14 want to adopt the proposed change for Annex J, item 57 option 2 from N3244 into C2y? 11-6-6
*Straw Poll*: Does WG14 want to adopt the proposed change for Annex J, item 57 option 1 from N3244 into C2y? 14-2-7 consensus to adopt option 1
*Straw Poll*: Does WG14 want to adopt the proposed change for Annex J, item 63 from N3244 into C2y? 8-3-10 not consensus
*Straw Poll*: Does WG14 want to adopt the proposed change for Annex J, item 67 from N3244 into C2y? 17-1-4 consensus
*Straw Poll*: Does WG14 want to adopt the proposed change for Annex J, item 69 using the Alternative Wording on Semantic Section from N3244 into C2y? 17-0-6 consensus
*Straw Poll*: Would WG14 like to see something along the lines of N3210 in C2y? 10/4/10 passed
*Straw Poll*: Would WG14 like to see something along the lines af n3211 into c2y Passed

### 8.2 Review of Action Items

*Action Item*: *Douglas*: Write a similar proposal to [N3247 using wording from Annex K.
*Action Item*: *Seacord*: Write a paper to clarify implementation-defined vs. unspecified behavior; do they require the standard to document choices?
*Action Item*: *Ballman*: Add more clear wording about semantics to address *Myers*'s comments to N3259.
*Action Item*: Update committee website for N3273 being on the extensions to obsolete versions of C list.

## 9. Thanks to Host

## 10. Adjournment (motion)

*Ballman*: Moved
*Svoboda*: Seconded

# End