Draft Minutes for 03 August – 07 August, 2020

MEETING OF ISO/IEC JTC 1/SC 22/WG 14
WG 14 / N 2581

Dates and Times:

3 August, 2020 12:30 – 16:00 UTC
4 August, 2020 12:30 – 16:00 UTC
5 August, 2020 12:30 – 16:00 UTC
6 August, 2020 12:30 – 16:00 UTC
7 August, 2020 12:30 – 16:00 UTC

Meeting Location

Teleconference

1. Opening Activities

1.1 Opening comments (Keaton)

Keaton: We have learned since the last meeting of the unfortunate passing of Walter Banks (Bytecraft, Canada), on December 9, 2019. He was a major contributor to Embedded C, TR 18037.

1.2 Introduction of participants/roll call

Etienne Alepins		Canada	
Aaron Bachmann		Austria	Invited guest
Roberto Bagnara	University of Parma	Italy	Italy NB
Aaron Ballman	Self	USA	WG21 liaison
Andrew Banks	LDRA Ltd.	UK	MISRA liaison
Rajan Bhakta	IBM	Canada	PL22.11 Chair
Melanie Blower	Intel	USA	
Alex Gilding	Perforce / Programming Research Ltd.	USA	Recording Secretary
Jens Gustedt	INRIA	France	
Barry Hedquist	Perennial	USA	PL22.11 IR
Tommy Hoffner	Intel	USA	
David Keaton	Keaton Consulting	USA	Convener
Philipp Krause	Albert-Ludwigs-Universit	Germany	
JeanHeyd Meneide	Self	Netherlands	
Joseph Myers	CodeSourcery / Siemens	UK	
Miguel Ojeda		Spain	Invited guest
Clive Pygott	LDRA Inc.	USA	WG23 liaison
Robert Seacord	NCC Group	USA	
Martin Sebor	IBM	USA	

Peter Sewell	University of Cambridge	UK	Memory Model SG
Nick Stoughton	USENIX, ISO/IEC JTC 1		SC 22 OR
David Svoboda	CERT/SEI/CMU	USA	
Fred Tydeman	Tydeman Consulting	USA	PL22.11 Vice Chair
Martin Uecker	University of Goettingen	Germany	
Freek Wiedijk	Plum Hall	USA	
Michael Wong		Canada	WG21 liaison

1.3 Procedures for this meeting (Keaton)

Keaton: procedure is as usual, taking straw polls in which anyone can vote.

David Keaton is the meeting chair.
Alex Gilding is the recording secretary.

1.4 JTC 1 required reading

The codes of conduct were observed.
Keaton: We avoid generating heat with no light.

1.5 Approval of previous minutes [N 2519]

[PL2211 motion, WG 14 motion] by Tydeman, Bhakta. Approved.

1.6 Review of action items and resolutions

Ballman: to produce wording documenting WG14 procedures for new members.

- OPEN

Uecker: to investigate a GitLab hosting solution at the University of Goettingen.

- DONE

Ballman: to study the character encoding type for string literals from `#error`, `static_assert`, and `nodiscard`.

- OPEN

Ballman: to submit a follow-up to n2480 with updated footnote.

- DONE (n2510)

Seacord: to submit a paper that proposes just the `%w` conversion specifier from n2465.

- DONE (n2511)

Svoboda: to submit a clarification request for C17 s3.4.3p3 (as outlined in n2466 p1).

- DONE

1.7 Approval of agenda [N 2538]

[PL2211 motion, WG 14 motion] by Bhakta, Tydeman. Approved.

1.8 Identify national bodies sending experts

Canada, France, Germany, Netherlands, UK, USA

1.9 INCITS antitrust guidelines and patent policy

The antitrust policy was observed without discussion.

1.10 INCITS official designated member/alternate information

2. Reports on liaison activities

2.1 ISO, IEC, JTC 1, SC 22

2.1.1 Recent changes to the Directives

Keaton: There is a major change to the ISO directives. The relevant national bodies must now be notified when someone is invited as a guest and may dispute the invitation, which may impose a chilling effect. Last year codified that guests can be invited. There may be an impact if we try to invite large numbers of WG21 members.

Ballman: how do we resolve a dispute, and how much notice do we give?

Keaton: invitations are sent as soon as possible. National bodies can respond right away, but might respond more slowly.

Gustedt: Why does this impact WG21? Can all SC22 members not attend all meetings?

Keaton: This was an unwritten rule, but was accidentally prohibited in 2017 and was not fixed in 2019.

Bhakta: does this apply to virtual meetings?

Keaton: yes, but there are no travel arrangements to disrupt.

2.2 PL22.11/WG 14

2.2.1 Document system

Keaton: The C-number system has been delayed by Covid-19 increasing Dan's workload. The ISO document system still has bugs, which breaks some N-documents. JeanHeyd Meneide is writing new software for the C-document system during the delay.

2.2.2 Optionally marking some future papers that clarify or fix text as "may be of interest to readers of prior editions of the standard"

In response to the reflector discussion there was a proposal to mark some papers as “maybe of interest” to readers of prior Standards, because WG21 wanted to know if changes are retroactive. Old Standards are withdrawn, not maintained, but topics can still be “interesting”.

Ballman: thank you – as an implementor, customers care about old Standards. Can we capture topics from the meeting discussion? It may be that paper authors don't think like implementors.

Gilding: volunteer to maintain this list.

2.2.3 Outreach

Keaton: this is made more important by the tightened guest rules.

Ballman: suggest an “incubator” group for members, non-members and champions.

Gustedt: maybe one big group for every theme?

Krause: what is the advantage over an n-document or a newsgroup post?

Keaton: we are not having internal debates in an external space – we can have an internal space with external posters.

Pygott: can we use the ACCU for this?

Keaton: that's external but worth making contact to advise them to look at the n-documents.

Ballman: who are the non-members this is for? Anyone can apply to join a national body.

Keaton: yes, if they want ongoing participation, but less effective for one-shot papers. Membership costs money.

Seacord: can a paper be developed in any context, outside the process?

Keaton: yes.

Ballman: We do have many one-shot papers, and this would make it easier for WG21 authors to port ideas over, and other people who need access to the committee without being members (a product of the new rules). This may also steer people into joining the group. Papers are rarely resolved positively in just one meeting. There are a number of areas where WG14 and WG21 differ on a common objective: could there be a formal common group? Though this sounds like unethically bypassing the ISO rules. A JWG needs an SC resolution, and WG documents can't be shared outside the WG.

Keaton: ISO n-documents are private; for JTC-1 *most* n-documents are public, and this is not problematic.

Krause: participation is not just about money. For instance there aren't enough people to mirror a group.

Gustedt: some people are affiliated to more than one standards body; this is more inclusive to such experts.

Bhakta: goal is good but we mustn't violate ISO rules by avoiding the NBs.

Ballman: ISO has made it difficult to participate with the constituency.

Keaton: request further debate and reconsideration next meeting.

2.3 PL22.16/WG 21

Ballman: WG21 has been holding weekly update calls instead of meetings, I haven't attended. Process is not yet fully coordinated. Work is progressing, and the mailings are now monthly instead of per-meeting, providing faster turnaround.

Gustedt: a shared calendar already exists; I have found the weeklies very focused and efficient, fixing problems well.

Keaton: two-weekly is difficult, need more focus.

2.4 PL22

Keaton: unhappy that we have such short notice on virtualization, making travel plans hard. ISO prolongs for three-month blocks. Want a better rule.

2.5 WG 23

Pygott: The C report is published, as well as Ada and Language Independent. New work on parallelism and OOP in the Independent report; not looking at C right now. C++ is active and working with SG12, two weeks per topic in weekly short meetings.

2.6 MISRA C

Banks: no change since the last meeting, nothing to report. We continue to progress C18.

2.7 Other Liaison Activities

None.

3. Reports from Study Groups

3.1 C Floating Point activity report

Bhakta: interchange types are now in the Annex. DRs are open on C2x, papers in this meeting and using the 2019 floating point standard.

3.2 C Safety and Security Rules Study Group

Pygott: the group is fading out, no recent calls. There are few active members, and the objective is unclear.

Keaton: Coronavirus and Charles's job change merit leeway, but the group must progress or disband.

Pygott: we have no Plan B after the failure to merge with MISRA.

3.3 C Memory Object Model Study Group

Sewell: We agreed to push a TS, which is drafted and sent. We encourage technical commentary and review by the WG. We suggest intervening meetings.

Keaton: I will send the required elements of a TS. I will ask for an 8-week work item ballot at the SC22 plenary August.

Sewell: Hand-count of interested members?

(Ballman, Gilding, Gustedt, Krause, Seacord, Uecker, Wiedijk)

The focused meeting to discuss end-zap with WG21 went well.

4. Future Meetings

4.1 Future Meeting Schedule

- 12-16 October, 2020 – Virtual, 13:00-16:30 UTC each day
- 30 November - 4 December, 2020 – Virtual, 14:00-17:30 UTC each day
- Spring, 2021 – Strasbourg, France (tentative)
- 4-8 October, 2021 – Minneapolis, Minnesota, US (tentative)
- 31 January - 4 February, 2022 – Portland, Oregon, US (tentative)

Keaton: Meetings are virtual to the end of the year. There will be two meetings to replace Minneapolis. A decision about Strasbourg will be made in October.

Seacord: can virtual meetings have a location for business purposes?

Keaton: yes, this is “Freiburg 2”. The mailing schedule has been reduced.

5. Document Review

Monday

5.1 Alepins, Unsequenced functions [n2539]

Alepins: to standardize a very old idea, 20 y.o. ; for better optimization, and for better safety ; introduces five properties, cumulative over each other ; two are equivalent to GCC attributes ; intentionally not yet applied to the standard library – need to consider floating-point state, `errno`, etc.

Seacord: why does `noleak` refer only to memory and not other resources?

Ballman: Files, locks, sockets, etc.

Gilding: How would this interop with C++'s `constexpr/consteval`? Is this intended to lead towards introducing those features to C?

Alepins: haven't considered the C++ interop at all, proposed without awareness of that feature.

Ballman: remember that these are attributes, can't change C's constant expression rules.

Gustedt: `constexpr` requires more than just `unsequenced` ; other leaking resources have other attributes? (to Seacord) other resources always touch some other global object state, so they are considered covered. n2522 adds more ways to describe the parts of state.

Myers: Isn't `noleak` a good property for any function? All functions shouldn't leak. Does the specification require a particular format?

Gustedt: Ideally, yes, but we cannot assume this because it is an allowed behaviour right now.

Svoboda: `realloc` etc. can “leak” in their arguments.

Ballman: These attributes are *really* useful in practice, but the proposal has a huge implementation burden: implicitly marking functions is too expansive: can't infer properties from a declaration only; can't infer properties for a function making external calls – proposes attributes which are “viral”, recursively propagating. There is a ton of good field experience with `pure` and `const`, but none with the other three.

Gustedt: the *intent* of the paper is to only be “viral” if you can see a definition – to deduce properties mainly for small inline functions. Aim is to be easy to use for application programmers.

Ballman: Great when the definition is visible; unusable when e.g. system headers are involved (e.g. applying `noleak` to a pool allocator built atop `VirtualAlloc`/`VirtualFree`).

Gustedt: there is a mechanism to define for C library; for 3rd-party code, will have to have an override mechanism to make it usable – puts verification burden on the user.

Uecker: recursiveness – attributes are not part of a function type, so pointer calls obscure this, e.g. a `noleak` function calling a pointer to a `noleak` function.

Alepins: We might need two kinds of attribute? Compiler-checked, and “assume”; pointers are not addressed in the paper.

Gilding: allow appertaining these to pointer parameters too? This will make visible on the caller-side how it will be used (or, if it will be called at all).

Ballman: I prefer flow analysis to a fully static pass, but this would cripple compiler performance by requiring dataflow analysis just to type check!

Gustedt: the check stays static if the property is “imposed”.

Alepins: what is the compiler burden? It needs to analyze this anyway for optimization.

Ballman: the problem is the same, should be solved both ways. Don't want the annotated library to impose long compile times – this is what needs field experience. Correct compilation can't rely on “implementation heroics”.

Bhakta: didn't like the “have to diagnose” requirement.

Alepins: will produce a new paper. Further discussion on the reflector.

Straw Poll: would the Committee like to see something along the lines of n2539 in C2x?

Result: 15-0-1

5.2 Ballman, Querying attribute support [n2481]

Ballman: this is returned from Ithaca: WG21 needed to allow changes to C++ to make possible ; C++ has added `__has_cpp_attribute`, but the question is the same and so will be the answer (guaranteed by standard). National Body question raised to change this before C++20, but WG21 rejected because they wanted to allow the same name to have different semantics; considered silly from an implementation perspective. There are no normative changes to the paper.

Answer is unsatisfying and a good example of non-collaboration ; may make changes at a directional level to take WG14 comments more seriously ; Direction Group agreed that this is a visible failure, but C++20 has shipped and the name cannot change. `__has_c_attribute` also already exists in the wild.

Bhakta: did the DG look amenable to a future common feature? (`__has_attribute`)

Ballman: EWG said no, but social factors affect this; there is some sentiment to support. We can't use `__has_attribute`, could use another name.

Banks: anything beginning __ is reserved? What is the risk of adding it? Prefer a non-specific query to force change on the C++ side.

Ballman: be careful not to break C++!

Myers: do the dates need updates?

Ballman: Yes, and two predicates allow for different results for both languages, which is one advantage. Otherwise need to use macro names.

Gilding: Can we additionally specify `__has_cpp_attribute` in C? Precedent of forbidding `__cplusplus`.

Ballman: maybe with a recommended practice, but that's novel.

Wong: for context – agree that divergence is part of a process problem – sometimes divergence is for good reasons, but features pass one way or the other with little time to react – most go back and forth, fall out of sync, and slip in the cracks; propose common meetings to agree on the core, have that back and forth at co-located meetings.

Ballman: then we need to schedule a common meeting.

Keaton: add a new agenda item, **7.5 C & C++ common issues**.

Ballman: `__has_cpp_attribute` exists; `__has_c_attribute` exists at least in Clang, but could be renamed; the fact that different dates are returned is a “sharp edge for users”; separating the names at least forces users to consider the dates may be out of sync.

Krause: `__has_attribute` isn't a problem?

Ballman: yes it is. Implementation experience shows that this doesn't work – GNU attributes work differently and the same name can have different semantics between contexts.

Krause: the answer is still “true”. Problem only for implementations supporting only one style.

Ballman: this only provides information for one full set – it doesn't tell me the standard attribute itself is available. Usage is affected as well as syntax, and there has to be a hard division between the two features.

Banks: what about `__has_std_attribute`?

Krause: `__has_std_attribute` looks as if it doesn't apply to implementation-specific attributes. Actually, GCC documentation states "__has_cpp_attribute (operand) is equivalent to __has_attribute (operand) except that when operand designates a supported standard attribute it evaluates to an integer constant of the form YYYYMM indicating the year and month when the attribute was first introduced into the C++ standard."

Keaton: If WG14 had originated it, this wouldn't be OK – but we don't have control of the situation. Need to continue liaising to converge the two features.

Ojeda: +1 Keaton's suggestion, there are already too many similar attributes when one works around C, C++ and several vendors. Any effort to simplify/converge is appreciated.

Ballman: are we comfortable maybe not resolving them? Joint meetings would keep the coordination common. We need a process to avoid accidental future divergences.

Keaton: because we don't have control, the divergence remains if we don't.

Krause: GCC documents them as the same except for the return value; implies not such a bad problem in practice.

Keaton: let's discuss the process now, before voting.

7.5 C & C++ common issues

Keaton: we know there are problems with getting a company to send delegates to multiple meetings – a virtual SG could help mitigate this.

Seacord: a dedicated study group is better than co-locating WG meetings, to force the agenda.

Wong: definitely issues co-locating – the size of C++ makes difficult; C spinning off SGs similarly to C++ (safety etc. to help MISRA), C & C++ on the same SG, like memory model SG. Serial co-location worked better in the past; teleconference needed anyway for triage and maybe ¼ co-located for dedicated collaboration work. There is a history of “bungled” issues as changes go back and forth, already between SGs. This is the latest casualty of a slow process, needs to be faster or divergence is inevitable.

Ballman: what's a JWG?

Banks: experience at SC7 – formal liaison committees, by SC; tend to be delegated power for a specific scope, e.g. the metrics group; C Core could be a subordinated scope for a JWG

Ballman: this is common to n2522.

Keaton: this is addressed by ISO directive 1.12.6: only possible between SCs, not WGs. We *can* create a JSG, as study groups are unofficial.

Ballman: want a group with authority so that the Core group can rein in the language-specific interests; it impacts the C++ ecosystem a lot if C diverges, C less so – lack of visibility could lead to a decision with a big financial impact.

Gustedt: each WG should create a study group consisting of the same members as the other and always co-locate their meetings. Authority rests with the individual members.

Keaton: on authority – historically, conveners dislike gratuitous differences (charter commitment to compatibility).

Ballman: C++ has no charter to be compatible – has the DG ever considered a charter?

Wong: actually, both groups are at fault. The charter doesn't prevent C from making changes anyway; a co-located group fixes real-time feedback and communication, which the charter doesn't address. I really push for any format of integrated meeting, occurring 1-2 times per year, as a common SG.

Keaton: physical meetings remain problematic, but virtual might work.

Myers: We could also make more use of the liaison@lists.isocpp.org mailing list that's existed since May 2019.

Ballman: I agree, it's not just a C++ problem. Bringing in attributes illustrated the feedback cycle problem. A charter provides instructions which a chair can refer to in discussion, guide discussion and consistently apply to both groups; would be a cultural addition like CWG.

Seacord: additional costs to a new WG vs. free participation in a SG. This will not be received well.

Keaton: that's national-body specific.

Banks: what costs? BSI doesn't charge for membership.

(discussion reveals that Commonwealth countries do not charge for membership, others do)

Wong: a joint SG that meets virtually, leaving open the door for face-to-face; having a charter doesn't *hurt*... but must have the same chair.

Keaton: what about co-chairs?

Gustedt: there should be just one.

Action Item: David Keaton to coordinate with Herb Sutter on establishing co-located study groups between WG14 and WG21.

5.2 Ballman, Querying attribute support [n2481]

Straw Poll: does the Committee wish to adopt n2481 into C2x as-is?

Result: 16-1-1

Paper will go on the queue for the new SG.

Myers: with editorial updates to the dates?

Ballman: yes. Will submit fixes and go in by next meeting.

5.3 Ballman, Minor attribute wording cleanups [n2527]

Ballman: this is based on implementor feedback and WG21 differences. Clarifies name spaces (the C term); provides wording symmetry with C++; removes unintentional restrictions on the syntax; unintentional other omissions (fallthrough); `for` is still asymmetric, but applies to both languages – this is an unintentional oversight; editorial change still needed for `opt`. Omnibus paper for general compatibility and consistency.

Propose voting on the non-`for` content first and to bring that later.

Gilding: what's the asymmetry?

Ballman: unintentional – thought this was already in C++, will propose the change there as well, then bring back with consensus from WG21.

Pygott: so this is cyclic..?

Straw Poll: would the Committee like to adopt n2527 as-is except for the section titled "Allow attributes on an expression in the clause-1 position of a for loop" into C2x?

Result: 17-0-1

Gustedt: can we have a direction for the `for`-loop?

Straw Poll: would the Committee like to adopt something along the lines of the for-loop proposal from n2527 into C2x?

Result: 12-1-4

Bhakta: I voted "no" because I want to see more before agreeing – that wasn't a hard-no.

Tuesday

5.4 Ballman, Unclear type relationship between a format specifier and its argument [n2483]

Ballman: This was last seen in Ithaca; the current wording is that the behaviour is UB if the "correct type" isn't used. This is ambiguous – can `_Bool` ever be correct? Should rewrite in terms of `va_arg` promoted type, which needs to match. Now uses the wording from `stdarg.h` etc., added to `fprintf`, `scanf` and friends. Seacord suggested some editorial changes – bullet points assume a "next argument", which needs to split out; only major change.

Bhakta: `fprintf` "will be stored into"? You can't store into an integer, only into an object. This wording doesn't actually say "object" or "type", though it implies it.

Keaton: you can't store into a "type", only an object.

Ballman: agree. We should lift into its own description for formatted input/output rather than describe it in four separate places, but not right now.

Bhakta: this isn't an editorial change, but "along the lines" is OK.

Ballman: we already got direction in Ithaca.

Gustedt: I think this is an editorial change.

Bhakta: I trust Jens, retract the objection.

Keaton: we can vote, but we need exact wording anyway.

Ballman: do we want "if there is no next argument" split into two? This could have an impact.

Seacord: I think it would be clearer, but not different.

Ballman: the rationale to clarify the bullets mentions types.

Wiedijk: type was already mentioned; the previous sentence needs to change or be moved since it's meaningless if there is no argument, though the current structure is OK.

Bhakta: another change of mind – we need new text.

Ballman: I will bring this back later.

5.5 Ballman, What we think we reserve [n2493]

Ballman: Also returning from Ithaca. We carve out reserved identifiers in awesome and terrible ways. Reservations using underscore or underscore-capital are well-understood, but pattern-matching on `is`, `to` etc. is not. `toilet` is *runtime* UB, because we *might* use it. All existing library identifiers are already explicitly reserved – this is too heavy a hammer for safety-critical code.

Proposal to introduce informatively-reserved patterns that are only actually reserved when used. We can add a recommended practice without needing to use UB.

Thousands of vanilla-English words are banned and no tool reports it all by default, so this helps toolmakers; users ignore the blanket bans anyway.

Krause: Not really counter-intuitive that according to 7.31.19 in N2479, `toilet` is reserved for use by `wctype.h`.

Myers: reservation is also mentioned in headers, not just the Future Library Directions. There are implementation-defined areas in e.g. `float.h`.

Ballman: hadn't included this in the survey.

Myers: it's not clear that all prefixes are equal - `str`, `is`, `to`, `mem` but not `f_` etc.

Ballman: some are also reserved and in use for compiler extension spaces – will need to examine them all and determine the classification.

Straw Poll: does the group want to see progress in the direction described by n2493 to include informatively-reserved identifiers into C2x?

Result: 12-1-2

Krause: can we have fewer reservations as well? We could cull some patterns? Reserving a lot is a problem, and we introduce new things in other patterns anyway, e.g. the floating-point namespace explosion.

Stoughton: POSIX reserves strictly by header, only if included. `stdlib.h` and `string.h` cause the most problems.

Ballman: in Ithaca some said, the reservation applies even in a different TU, as the external identifier will be linked. POSIX doesn't fix this; also since any identifier can be a macro, the preprocessor-level is also reserved.

Myers: new symbols will inevitably fall outside these. Patterns are for clear categories and can't generally cover everything. It will always be possible to require a diagnostic on a fixed list of names with external linkage.

Ballman: we could state that the list is not exhaustive.

Krause: non-exhaustive notwithstanding, can we have a list of non-reserved identifiers guaranteed to work? Standard functions are allowed to be used without including the corresponding header, which breaks the POSIX idea.

Gustedt: I like the paper, but the wording is really missing. The second aspect is that we also would have to stick to these rules when we actually do new things.

Ballman: that raises the question about intent, and whether we intend the patterns to be exhaustive. (to Keaton) Was the pattern means to be a promise?

Keaton: I wouldn't agree with the "whole bunch", but a good chance that *most* will come from the list.

Ballman: it is non-exhaustive, then.

Seacord: I would say no, these were reserved for certain purposes.

Gustedt: this also provides implementation convenience, not just future expansion? e.g. additional `errno` values that don't need to check whether POSIX is enabled or not to be made available.

Conditional reservation existed in C90 for `wchar.h`/`wctype.h` on header inclusion – why was that rule removed in C99?

Krause: implementors reserve all external identifiers unconditionally (7.1.3)

Gilding: this introduces an item to the “of interest to users of obsolete standards” list.

Keaton: right, but it's *not* retroactive!

Ballman: I will bring back a new paper separating identifiers.

5.6 Bachmann, Make pointer type casting useful without negatively impacting performance [n2484]

Bachmann: this proposes to *re*-allow pointer casting and access in a local scope – there is no global effect, and the strict-aliasing rule continues to apply. It is important to be able to avoid assembly and still be able to use bigger-than-`char` copies ; this is important in the embedded domain ; standardizing C should allow for users to write core operations in C ; prior attempts were more radical, this tries to keep performance advantages and doesn't disable type rules.

Wiedijk: what about type punning with unions? Can this replace casting?

Bachmann: no, because objects have a declared type which we can't change.

Krause: the implementation already knows if it's safe to use a coarser granularity than bytes. Efficiency should already be possible because optimizers fuse byte copies.

Bachmann: GCC still often can't do this in practice.

Myers: I don't think the wording works – the pointer conversions don't define that the result of an object-pointer conversion is actually usable for read/write, only that it will round-trip. Implementations give warnings for the most obvious cases of bad pointer conversions, which should help new users know about the issue. It's problematic for implementations to talk about block scope because optimization happens after scopes are messed up by inlining. We need a `reinterpret_cast` operator to explicitly mark l-values as being re-typed, not pointers at all, and impossible to alias.

Bachmann: so better to use a keyword, like `restrict`? I'll try that if we proceed.

Gustedt: I like the *idea* (needed), but you should coordinate with the memory object model group who do a lot on reinterpreting. n2522 also has an attribute for this, to re-type for the duration of a call – coordinate here too?

Bachmann: does this help? You need to “peel off” the head and tail for e.g. `memset`, when the alignment is less than the word size, and use smaller accesses to avoid off-the-end writes.

Gustedt: there are lots of applications for this (e.g. matrix to vector), but it should not be unrestricted. In n2522 this is restricted to types with the same representation.

Bachmann: I will establish contact with the memory object model group and completely rewrite the wording. Rules for a user should not be too complicated – the SG's wording rules are not easy for users, and the language is for users.

Gustedt: the intent of the memory object model group is not to overcomplicate! but to generate wording.

Uecker: I like the direction, but also want to send to the memory object model group, as this impact effective types and so on. We should vote on the direction anyway.

Straw Poll: does the Committee want something along the lines of n2484 casting added to C2x?

Result: 6-3-6 (no consensus)

(abstainers want to hear from the memory object model group first)

Seacord: I'm voting no because I prefer the `reinterpret_cast` which I'm interpreting as being along different lines.

Myers: Likewise, I'd like a way of doing such accesses but not this one.

Ballman: Yeah, I generally like the approach taken by the C++ named casts.

Bhakta: I like changing type via cast, but is there direction on which way?

Bachmann: this would be for lvalues – pointers are just a way to convey this.

5.7 Bachmann, Allow memory-reuse via pointer-casting [n2537]

Bachmann: this complements the previous proposal, and would allow for the implementation of `malloc`, memory pools, etc. directly in C. This is for global scope. It may apply to memory allocated during before-`main`; we do not lose the effective-type rules by inverting the `char`-access rule and allowing other types to alias `char`. Observing real program behaviour, there is no aliasing because accesses are sequenced by type. There is no conflict, because `malloc` doesn't reuse the memory until after `free`.

Myers: C++ experts need to weigh in on whether this conflicts with the rules for placement `new`. It must not be incompatible.

Gustedt: exactly – this redefines the effective type – a good idea, but something handled by the memory object model SG, who are also converging on a placement-`new` equivalent – this is not trivial to integrate into the memory object model.

Bachmann: I will get in touch with the memory object model SG.

Ballman: I like the direction.

Straw Poll: does the Committee want something along the lines of the memory reuse described in n2537 to be added to C2x?

Result: 9-1-6 (direction yes)

5.8 Bachmann, Add explicit_memset() as non-optional part of C2X [n2485]

Bachmann: (there is an error in the paper: musl doesn't have this function)

There is a need to erase memory in secure contexts ; the standard library should provide it, but prior work differs ; the signature of `memset` is well-known ; the proposed name is based on common usage.

Gustedt: this is a good idea; I prefer the feature of non-zero “poisoning” of cleared memory; prefer the `memset_explicit` name pattern though because it fits with existing namespace reservations. Maybe the signature should add `volatile`?

Krause: We can't just stick to names already in use. We reserve names for future use. People come up with functions, avoiding reserved names. We promote functions into the standard. By sticking with established names, we get exactly those identifiers that are not reserved.

Seacord: I was the original author of one such proposal, which was pushed to Annex K; the obvious solution (barring the minor dependency issue on other Annex K features) is to pull `memset_s` and its dependencies up from Annex K? These are already in use. It was wrong to force duplication on Annex K implementors.

Ballman: the wording lifted from Annex K about "strictly according the rules of the abstract machine" - does this achieve the intended goal? 5.1.2.3 has "needed side effect", which ensure that the call cannot be removed.

Svoboda: I presented `memset_s` and didn't know about 5.1.2.3! We couldn't really formalize out-of-scope technologies like cache, disk, etc. in the Standard. `memset_s` doesn't use handlers, and can be transferred; changing `errno_t` is a trivial transformation.

Bachmann: `memset_s` is suitable for single-threaded execution only.

Seacord: the multithreaded problem is solved elsewhere.

Ballman: would Annex K remain optional? Can't really rely on it being available.

Seacord: rather than duplicating functionality, we should just promote the most popular functions and their support mechanisms to the main Standard as non-optional.

Svoboda: separate papers needed for `errno_t` etc. will need to cover the interactions.

Seacord: a mechanisms paper can be written first – this isn't a rabbit-hole, just the features that are actually needed.

Sebor: no objection to the API, but the wording of the last sentence implies the object remains accessible, by requiring that the values stay in the object. I am opposed to moving anything up from Annex K, especially constraint handlers – these are problematic for a security-oriented API. I prefer an established name and signature – changing it is confusing to new adopters and legacy maintainers. Tweaking a signature is error-prone, maintaining existing practice is safer.

Bachmann: this isn't an invented name – there is practice.

Sebor: the signature differing would be a concern for usability, and should be consistent. I don't see how the wording provides any guarantees – it has no teeth.

Wiedijk: agree that this is like `volatile`, the wording is not adequate – the as-if rule allows the write to be omitted; I favour something like this but don't know how to specify it acceptably.

Svoboda: the last sentence, "shall assume", comes from `memset_s`, which may give the impression that memory *must* be accessible with values – we wanted to express that the old value is gone, not available for future reads, so it should say "does not" contain, though maybe this implies indeterminate content.

Tydeman: would a "must execute" attribute help?

Ballman: wouldn't be needed if the functionality is specified correctly and doesn't add anything to the clear expression of the behaviour.

Myers: A better use for an attribute might be on a function as a hint to clear memory not directly accessible as a C object at all (see the brief reference to that kind of thing in the next paper).

Ballman: The problem with an attribute is that ignoring the attribute would change the semantics of the program because you could observe that effect. Sort of observe that effect, maybe?

Svoboda: The attribute would also have the same problem with the wording that `memset_s` has. It might be useful if you want to use it for other functions too.

Myers: A stack-clearing attribute would be a hint about what the implementation should do with things not accessible by valid C code at all.

Ballman: I still think "needed side effect" is the right specification approach as we already talk about that in the abstract machine section.

Myers: The whole point of this sort of thing is that an attacker might subsequently gain read access to the program's memory (i.e. what happens after subsequent undefined behaviour, or what's visible by means not described in the standard, since e.g. speculative execution attacks might not involve undefined behaviour.)

Krause: I don't think we'll easily find a better wording than stating that the writes happen as if for volatile.

Wiedijk: I think the wording really should contain the words "side effect", as introduced in 5.1.2.3.

Svoboda: The next paper has some discussion about better wording. I think we should table wording discussion until we've reviewed the next paper.

Gilding: cache and disk *are* a problem, and we need to express that somehow, otherwise the user might trust the function to do more than it actually does.

Gustedt: we can't act on anything outside the abstract machine scope, which is what we have. Wording along the same lines as `volatile`, which should also be in the signature as it will probably need to accept `volatile` operands anyway. Also link to Niall's paper about cache. Recommended practice is the tool to express cache and other out of scope issues.

Pygott: we can't specify as “can't be the old value” because the old value might be rewritten.

Svoboda: intent is needed here.

Seacord: the abstract machine language from `volatile` was a way to express security, but the idea that anything imperfect isn't secure isn't what `_s` means. Nothing is perfect and we can't dismiss this because it's not perfect, that prevents progress. This is incremental progress and we can address other aspects separately.

Gilding: my concern is that the user would trust the function too much, but a footnote telling them not to could work.

Bachmann: I would rephrase with `volatile` qualification and base the wording on the `volatile` qualifier.

Keaton: we'll compare and contrast after seeing Miguel's paper.

5.9 Ojeda, secure_clear [n2505]

Ojeda: This feature is in use in the Linux kernel and elsewhere. Implementations are buggy and depend on optimizers. This was already discussed in WG21 by LWG and SG1, where it was positively received. C++ were concerned about wording; wording is not part of this proposal, no specific implementation.

This differs by not accepting a value ; value is not important for this use case, only overwrite ; indeterminate, could be zero or something else. If the compiler knows there is nothing to erase, it mustn't make it less secure – hard to express correctly ; neutral on the caches etc. issue, memory object model SG want it to be recommended practice or QoI, to give freedom to implementors to do the things we can't express. Tend against `volatile` because in C++ this means memory must be touched ; WG21 didn't like `memset_s` wording; `volatile` wording may imply a set of writes vs. one write; how many should be specified. The intent is clear but specifying the side effect is movable by the optimizer and could be removed. Seeking direction to review wording with the memory object model SG. One option is to avoid abstract machine specification entirely and only convey intent in the recommended practice.

Differences from Aaron's proposal include coordination with C++ (and some C++ content to avoid mistakes); not specifying a value to write; coordinating with the LLVM team to ensure the proposal is implementable; and providing an option to avoid non-memory writes. This is not equivalent to the Annex K function.

Agree with Martin's concerns about the name, and the C++ committee is very concerned about mistakes – don't call it "secure" if it isn't, and don't encourage user-side errors – we want user clarity. C++ naming is more flexible in general than C naming though.

Krause: Having secure in the name might create a false sense of security. While the function would be about an aspect of security, other aspects probably got missed. Also, again I'd prefer a name from the reserved space. So memset_explicit, memclear, etc. seem like good names to me.

Gilding: is "indeterminate" the right word here? Though that could have a useful impact. The intent to avoid writes when a value is stored in a register – does this imply the value is allowed to remain there?

Ojeda: only a *memory* clear is expected – I hadn't considered a register clear, which is hard to describe in Standardese, but the value shouldn't be left there. "Call could be elided" is only supposed to imply that memory writes can be elided.

Uecker: This is one issue with all these functions, information could leak in registers, in the cache, on the stack, etc... I am not opposed to add some best effort function, but it will never be secure. A non-ignorable function property is needed to fix it.

Krause: In SDCC it would be hard to reliably clear registers, as the last stage is a peephole optimizer, which is likely to optimize out writes to registers that are not read again.

Gilding: additional overloads would also be an option in C, we can express that now. We can make the value optional, or force a constraint error on attempting to clear a pointer object. Either way we should be compatible with C++.

Svoboda: there was some debate on the wording for `memset_s` about ten years ago, but less than this. Where did "indeterminate" come from? C++?

Ojeda: it's original, but C++ liked it. The use case is targetted towards clearing, not setting, so the focus is off the value – we don't care what is there afterwards. Normally the object will not be reused as-is.

Svoboda: in that case a function that did nothing would be a valid implementation! Intent is important here because we don't have the terminology to make it exact – this depends on intent even more than `memset_s`.

Ojeda: The `memset_s` wording is only chosen as an example. The value is not important from a user perspective.

Svoboda: clearing registers is out of scope anyway, as you can't take their address.

Ojeda: varying opinions from implementors here on how do-able and relevant this is. AN attribute was discussed, but faces the removal problem and implementation issues. Stack-clear has little implementation experience and is not widely implemented. Providing memory clear is the minimum actually required by users – other solutions are more advanced and could work on them as they come up.

Svoboda: Given that we have two papers that 'fix' `memset_s`, I don't want to wait to fix Annex K. We should treat memset_s separately, for now.

Krause: Promoting `memset_s` to required would introduce a lot of Annex K baggage (`rsize_t`, runtime constraint handlers).

Svoboda: As it is, yes. but clearly we can separate `memset_s` from that baggage, as Aaron and Miguel have done.

Blower: If we have `volatile` on the function argument, does that mean the function call cannot be removed? By the optimizer?

Krause: But then it should no longer have the name `memset_s`, so instead of promoting `memset_s`, we'd accept N2485 or N2505, or something similar.

Myers: Constraint handlers are a problem because of action-at-a-distance (a library has no idea what effect an error in a function it calls might have), not just because of the thread-local issue.

Seacord: I prefer the `memset_s` signature to this and the previous proposal; maybe a clear, not a set? No requirement right now to overwrite with random data. I am opposed because we have the function in Annex K, and the problem is that the future of Annex K is unresolved. We have had discussion of experience and how to handle Annex K, but no consensus to remove in prior meetings. There are good ideas in Annex K, from Microsoft in particular – can we fix it? What changes would be needed to make Annex K required, and then `memset_s` will be brought along too, and fill the "gaping" void.

Ojeda: value vs. n-value is a good interface choice; opposed to making better in what we guarantee vs. what we permit. Can Annex K be fixed in a reasonable timescale? There's opposition to a lot of content.

Seacord: Martin Sebor brought an experience paper; I refuted it; the committee was split with abstentions.

Ojeda: we can do this ahead of Annex K and merge when Annex K is promoted to required.

Seacord: that's convoluted, and it's hard to remove content once it has been added. The solution to the single-threaded issue for instance depends on the fate of Annex K being resolved. Removing the problems implementors have would simplify the process.

Bachmann: writing a defined value is better than an unspecified one for the same effort. There could be multiple following code paths, and the next path may want to reuse memory with zero, which is better than duplicating the operation. On the reuse path we could have just one call.

Ojeda: this is about the user not caring – I don't think the performance matters. If you *are* reusing memory, set a sequence point and give the value normally? The bigger point is whether we want it in the interface, not whether it is zero.

Ballman: interesting signature focus on clearing rather than setting, I like the perspective. The issue with `memset_s` is that attackers can even learn something from that zeroed page – there is a security benefit in letting the implementation do a random scrub, so does setting the value potentially hurt security?

Ojeda: the use case is to tell the compiler to dispose of values and make them hard to recover. `memset_s` biased the proposal, but the purpose is disposal.

Gilding: is the implementation experience from Linux and elsewhere better than Annex K? What interfaces are preferred?

Ojeda: there's a list; most zero or support choosing a value, but the wide variety shows the different ways users want and use the feature.

Pygott: Common to both proposals is that they can't be optimized away – there was a prior proposal for a `defensive` attribute, which we rejected.

Gilding: That was for a different use context and was much harder to implement than on a function call boundary.

Bhakta: The `memset_s` use case relies on `volatile` in practice, and users assume that it works, which might be wrong.

Gustedt: I like unspecified values too, and the template-style interface would be able to do the register-clear – we could reformulate this to pass an object and destroy or zero it, similarly to the C++ interface.

Krause: `defensive` was to prevent the compiler from optimizing out reads and writes - `volatile` does this anyway. There were performance concerns, but hard to address in a better way than `volatile`. Don't want it to look like a new syntactic structure.

Gustedt: it would look like a macro call, and specify an action on an argument.

Keaton: we need a vote.

Bachmann: mine was first, but work on either Annex K or this paper would fulfil my goal, so I am willing to pull my submission.

Bhakta: do we have sentiment to do something? Not to promote from Annex K without an actual proposal.

Myers: vote on specified vs. unspecified values?

Ojeda: I want to separate the interface and the implementation. If we don't want unspecified values, then we should decide on a value afterwards.

Straw Poll: would the Committee like to see a non-elidable, non-optional memory-erasing function added to C2x?

Result: 14-0-2 (clear direction)

Straw Poll: would the Committee like the non-elidable, non-optional memory-erasing function not to specify a value in its interface?

Result: 6-5-6 (unclear direction)

Straw Poll: would the Committee like to be able to specify a value in the interface to the non-elidable, non-optional memory-erasing function?

Result: 7-4-6 (clearer direction)

Ojeda: are there other polls or results to consider? Because WG21 liked not having the value.

Keaton: no cross-committee polls.

Ballman: I abstained because there's already research here – I want this research to inform the decision.

Straw Poll: would the Committee like to have both no-value and value-specifying interfaces to the non-elidable, non-optional function available?

Result: 5-6-7

Keaton: this sounds like more discussion is needed.

Wiedijk: I do think this needs to be taken up with the memory object model study group too.

Ballman: FWIW, my abstain is because I'd like the paper to research the security implications of specifying a value vs not.

Stoughton: And for me, my votes were driven by the level of existing practice, strongly in favour of just clearing.

Seacord: just clearing is different then overwriting with random values. I should know this, but will overwriting with random values for example eliminate row-hammer and other side channel leaks?

Wednesday

5.10 Krause, register at file scope [n2486]

Krause: This proposal uses the “correct” meaning of the keyword – no-address, plus a hint. This is most useful to communicate across translation units; currently, the keyword is allowed where it is not needed, and not allowed where it would be useful. Globals are still used extensively for data passing on small systems. An attribute could do this, but the keyword already exists. Arrays are not considered by the proposal, which only moves the application of the keyword.

Gilding: does this consider C++ interop? `register` is removed as of C++17, it's just an identifier. This could impact header interoperability.

Krause: didn't know that... but it's still available in C.

Ballman: register in C++ is still reserved, which makes the edge a bit sharper.

Gustedt: what linkages are available for `register` objects?

Krause: the only change is that you can't take the address, so it adds to the storage duration rather than replacing it. The default linkage is external and it is UB if not consistently declared.

Gustedt: what about functions?

Krause: hadn't considered those.

Gustedt: will bring forward an attribute for this tomorrow anyway.

Myers: there's no obvious use for this with `static`, as the implementation can see all usages. It gives information when combined with `extern`, but this is still a basic LTO and fairly obsolete in that context.

Krause: most C compilers still don't support LTO, especially on microcontrollers which don't have C++'s investment. It's not realistic to expect that kind of compiler power for small targets.

Uecker: the attribute wouldn't be the same as a language feature? It can be ignored? It shouldn't be allowed to be ignored and require a diagnostic for inconsistent declaration. I therefore prefer a keyword. Since this removes an arbitrary restriction, it simplifies the language by allowing the keyword in more places and makes it easier to understand. We have the right to keep a feature regardless of what C++ does.

Krause: agree that an attribute relies entirely on QoI.

Ballman: I have the opposite take on attributes – they are trying to give information to the implementation, and a *correct* program can ignore them without changing semantics – this is not different from e.g. `maybe_unused`, `nodiscard`.

Krause: compare `const` - if you remove it, the program stays correct, but you can't ignore it. Argument either way but prefer existing keyword.

Myers: `register` no longer has anything to do with registers. An attribute would be more meaningful to users reading code, contrary to the language expectation to bind real registers, which has nothing to do with the name.

Seacord: how likely are we to remove the keyword? If not, we don't break so much.

Krause: it does get used in embedded code, we would need a good reason to break it, it's not unsafe and has wide usage.

Uecker: I like the name `register`! "Addressless storage" makes sense and is easy to explain.

Keaton: what is the impact on TR18037 "Embedded C"?

Krause: I haven't checked but did see that GCC implements this. Don't know that this feature is used much, not as much as pointer namespaces.

Straw Poll: would the Committee like to see something to specify that the address of a global cannot be taken?

Result: 8-0-9 (clear direction)

Straw Poll: would the Committee like to use the keyword `register` to prevent taking the address of a global?

Result: 8-6-3 (no consensus)

Straw Poll: would the Committee like to see an attribute that would mark an object as non-addressable?

Result: 12-5-1

Wiedijk: I would like the hint, but not taking the address is not the same as non-addressable.

Bhakta: I agree with Martin. Do we also want it for block scope, or two different features?

Ballman: Yes, I want it for block scope.

Gilding: Perhaps it's time to "obsolete but not remove" `register`, to avoid maintaining two specifications.

Keaton: I am wary of impacting the Embedded C feature.

Uecker: Why would we remove and replace an existing feature?

Ballman: Users don't understand it in practice, it is misused and misapplied. Rules ban it. An attribute lets us put a new clear name on the feature and fix the design.

Bhakta: This is catering to a small subset and not the wide usage in the field, pruning features not useful to some? My users are not confused, and widely use it.

Keaton: The paper does not propose removal.

Gustedt: Both have their own field of application – they aren't the same, one is much stronger, as the attribute doesn't require identical declarations. We shouldn't deprecate a widely-used feature.

Myers: this is useful for Embedded C, and should be kept there. Other usages are confusing and legacy usages. Most users don't care about addressability, only the hint, which is not useful in Standard C.

Uecker: I think it does – people use it correctly, to give hints and enforce non-addressability. Optimization is better if the address isn't taken, and the paper just expands existing use. This is its intended purpose and I disagree that it's misleading.

Krause: `register` in Embedded C allows naming a register and actually does use it – I don't see a conflict here, named vs. unnamed syntax.

5.11 Krause, short float [n2487]

Krause: This is a follow-up to Nvidia's 2016 proposal, which the committee liked, but Nvidia no longer wants it, or even `long double`! In the spirit of the original, this proposes minimum requirements but not a fixed representation. This is important in image processing and in machine learning, and for small systems without hardware floats, which are still the majority, where floats are emulated and need a cheaper format that is not tied to representation.

Uses for 8-bit floats are very rare, and it's not really needed, but we do want multiple representations.

Gilding: could we add both `short float` and `char float` for symmetry with integers? Have both available?

Krause: I don't like that syntax... 8-bit is really obscure, this proposal is mostly about adding 16-bit floats.

Bhakta: I agree we don't want the sign limit. In 2016 there was a lot of ambiguity – a lot of implementation experience since then shows support is gone: ML is much more abstracted and not language-specific now. The original motivators no longer exist. If software emulation is a burden, adding this increases that burden! Most people won't want this.

Krause: IAR, Cosmic etc. don't provide floats at all, or provide just one type. Providing the smaller type makes it cheap enough to be usable; defining a smaller type makes it even more cheap.

Ballman: there's an explosion of floating types, losing the thread of when to use which type. has the floating point group seen this?

Bhakta: we did discuss it, favoured IEEE in 2016 who later removed the arithmetic type, and kept the spec as an interchange format. `_Float16` fits into C2x like "the rest" in that model; it doesn't add cognitive load if you're already familiar with it.

Ballman: What about file I/O specifiers? Any other library changes?

Krause: These are intentionally not included yet, first checking if we want the type itself. Would a format specifier even work?

Ballman: yes, with length modifiers.

Myers: raised concerns on the reflector that code really *needs* to know the format details, and a generic type won't help. Annex F specifies how to provide some types but not others, would need separate listings for `bfloat16` and `_Float16`. In C99 there was a DR for complex floats in `va_args` - we didn't want to promote anything new in variadic argument lists.

Krause: It would be weird if `float` promoted and not `short float`? The types look different but should behave similarly to basic types and promote to double. The compiler can optimize the precision for math.

Hoeffner: users care about *every* bit. Without a guaranteed format, they wouldn't use it. Who is the target? The withdrawal by Nvidia indicates that.

Gustedt: if you're looking for floats with a lower bar, why not relax the requirements on `float`?

Krause: existing software relies on the guarantees of the old specification.

Myers: `bfloat16` adjusted `flt_round` to precision define modes; this applies to all types. Intel version doesn't work with it, `fe_setround` not affecting all modes, rounds to even only, while on ARM rounds to all: so `bfloat16` on hardware doesn't match the Standard's model.

Keaton: `bfloat16` is more useful for machine learning, IEEE is better for image processing and other things, so there's a different choice depending on the application.

Bhakta: as Tommy said, use cases know what they use every bit for. We can't make this work without a ton of macros and configuration, and the implementors don't even want it!

Krause: Get the impression the committee doesn't want this; I don't feel a vote is needed and will not pursue this.

5.12 Krause, accessing const objects from signal handlers v2 [n2523]

Krause: this is a normative follow-up to n1812. The rules are restrictive and there is no reason to forbid const reads. Implementation experience confirms.

Myers: There is still an issue even when “referring” to `_Thread_local` variables, which computes an address and may be in allocated memory. This might not actually be distinguishable from “access”.

Krause: OK, not an expert on thread local storage.

Sebor: I support the relaxation, modulo `_Thread_local`. But the original paper addressed the handler accessing locals (which wasn't prevented) – has that been fixed? The same problem exists with allocated objects – not prevented, missing constraints. Want to fix these gaps.

Krause: I don't see a problem with preventing this. This is much more common than e.g. reassembling an address via `uintptr_t` read from flags...

Sebor: I disagree: I don't want to relax constraints in isolation without tightening the others first.

Krause: a new revision is needed for the `_Thread_local` issue anyway, to present at the next meeting. A vote isn't needed, but I do want direction.

Sebor: I am willing to collaborate or address the issue separately.

Straw Poll: would the Committee like to allow access to non-thread-local const objects from signal handlers, along the lines of the proposal in n2523?

Result: 16-0-3 (clear direction)

Straw Poll: would the Committee like to allow signal handlers to refer to objects without accessing them, unless they are thread-local?

Result: 16-0-3 (clear direction)

5.13 Krause, string functions for freestanding implementations [n2524]

Krause: people want to use `string.h`, and usually have to reimplement it. This isn't hard, but it means the implementation can also provide it easily. Already in the Standard, easy to provide even on *tiny* targets. Implementors want to help users reimplementing it.

Gustedt: This adds a constraint to all freestanding compilers – which others support it?

Krause: Keil, IAR, Cosmic, Renesas all provide these functions.

Bhakta: remember that embedded is not the only client of freestanding!

Sebor: In favour, but concerned now that we adopted `strdup`, `strndup`, which allocate and need to be excluded; `strerror` might be problematic. `errno` isn't freestanding either.

Krause: `strerror` will usually just be a switch over literals on small implementations. We include `errno` in the paper. If there's no dynamic memory, the implementation could provide stubs that do nothing; SDCC sets up “dynamic” memory at compile time, the uses the preset pool, so it can provide it.

Bhakta: as above, freestanding includes mainframes. `strerror` is locale-specific, so it has to not be listed. Ours doesn't provide *any* memory management – this is the point of freestanding! The other calls, we ship and make sense/, but I strongly object to adding locale or allocating functions, this is not what freestanding is about.

Seacord: These functions have a not-great history with security, and now we spread that further.

Krause: These are what people already use, either by extension or by reimplementation. I didn't consider adding Annex K – looked at existing practice and standardize usage.

Ballman: Is the intent that if a user defines Annex K, they don't get it?

Krause: No, that's out of scope.

Straw Poll: would the Committee like to make string.h mandatory for freestanding implementations, minus the functions `strdup`, `strndup`, `strcoll`, `strxfrm`, `strerror`?

Result: 12-0-6 (clear direction)

Straw Poll: would the Committee like to see further proposals to make more of the Standard Library mandatory for freestanding implementations?

Result: 13-1-5

5.14 Krause, remove the fromfp, ufromfp, fromfpx, ufromfpx, and other intmax_t functions [n2525]

Krause: There are lagging references to `intmax_t` in the standard library: 24 new functions in C2x are using it, that the floating point group will fix, and two other places: `inttypes.h` and the `%j` specifier.

Bhakta: CFP does have an alternative to this: we want to delay until both can be compared, as the approaches are opposed.

Krause: CFP's alterations affect the first area and change it to use other types; the second and third areas are not addressed.

Myers: CFP's type change is the better approach; `printf` and `scanf` only changes half of the formatted-I/O issue; would it be user-friendlier to change `%j` to be an alternative for `long long` rather than remove it?

Krause: we can't just change the meaning, and `intmax_t` is gone.

Svoboda: we haven't actually removed it yet, just set the direction and are waiting for a paper. We should discuss that before this.

Seacord: I agree, the decision is exaggerated and there is no solution with consensus. I'm OK with removing the functions and leaving `%j` and the type for now.

Myers: a paper doing the removal should cover all the cases – preprocessor, FP functions, etc. - a coherent proposal for everything should come before a selective removal of parts.

Gustedt: we were told to make small changes and small papers? This makes progress, removing the functions is a first step to that.

Gilding: reiterating that, this is commitment to the removal, makes progress, and cleanly subsets the problem.

Krause: The ABI problem is addressed by removing parts of the ABI, which is the biggest obstacle and the type itself is a second step, and lesser problem; this also makes way for Melanie Blower's types.

Myers: on a case-by-case basis for each function, we should fix the ABI or remove it, for some change makes sense and for others removal. `%j` is user-friendly to leave, and allow use of `long long`. This raises the question of what to replace `intmax_t` with, rather than a global removal.

Krause: changing `%j` introduces danger by changing the meaning – supplying the wrong type after the change is now mismatched.

Svoboda: most conversation about `intmax_t` in general should focus on the larger question. The smaller paper has ramifications for the bigger problem, tipping it – if we didn't remove the type, are these three questions worthwhile?

Krause: yes because the problem is in the ABI. If we remove it here, the ABI problem is solved (if users don't use it). It doesn't solve the "huge type" problem, but it does solve the original.

Svoboda: so it eliminates a constraint giving us the freedom to modify the type.

Keaton: I have retrieved the CFP paper (which has no number yet) – we can consider it informatively.

Tydeman: there are two sets of functions: some returning `intmax_t` are converted to return the floating type, which improves functionality because they can now also return `NaN`. Those accepting an argument are converted to accept `long long`.

Bhakta: `intmax_t` was only used for consistency, not because we thought it was the right way to implement this.

Straw Poll: does the Committee want to remove the new `intmax_t` functions?

Result: 6-7-6 (no consensus to remove)

Straw Poll: does the Committee want to remove the old `intmax_t` functions?

Result: 5-5-9

Straw Poll: does the Committee want to remove the `%j` specifier from the `printf`/`scanf` family of functions?

Result: 6-9-4 (no consensus to remove)

Bhakta: for those voting Yes on 2 – is the sentiment to not follow IEEE? This is part of IEEE conformance.

Meneide: I view removal as a precursor to the next mailing paper, fix the ABI trap, prevents better ints from being used. We should be purging uses from the standard library to open the door for ext-ints that are already supported – anything stopping that impedes better int functionality. I would accept the CFP paper (remove and re-add).

Krause: we can always remove and re-add without `intmax_t`; I dislike the FP explosion and namespace pollution.

5.15 Thomas, C support for ISO/IEC 60559:2020 [n2531]

Bhakta: Informative, not seeking votes ; update on goals set by WG14. Goal was to get IEEE 754-2008 into the C Standard ; brought in as a whole, minus contradictory requirements ; optional stuff partly in, mandatory content mapped fairly well. 2020 release increased scope to support in C, bugfix release, no new features, everything new was optional; 2030 revision will make some mandatory. WG14 will only bring in a published draft. Small additions to C2x bumping from 2008 to 2019 with small changes. CFP did a really good job with WG14; 2008 incorporation influenced IEEE, who went beyond scope and added features back to IEEE; excellent cooperation between organizations and standards. RTTE – defect; min/max – seriously flawed, removed and replaced; WG14 wanted to keep old functions, hence new names for compatibility, which is permitted. New Quantum worked well bringing into C2x.

Feature tests and binding tables brought in as-is; C did not want reduction functions ; don't yet know if they will be mandatory, probably not now ; no proposal for augmented-add etc. ; proposal later for corrected min/max ; payload functions already there and no change needed.

Believe we want to point at the new IEEE draft, big version bump, sensible if we adopt the new content.

Myers: so IEEE will make `sin`/`cos`/`pi` required?

Bhakta: those are optional in 2020, were recommended, not required in 2008. Also very hard to implement in correct form as these specify correct rounding; binary-128 is too precise to implement practically. Might not be added as a correctly-rounded specification only, has been debate, theoretical side want all correctly-rounded, others call this impractical without overhead; expect to be status quo.

No start on the next draft yet though, or even a committee.

Myers: can WG14 get likely suggestions in advance to provide feedback?

Bhakta: yes if the CFP still exists at that time.

Tydeman: total-ordering in C2x differ from the TS, which affects at least one implementation; payloads also changed between the TS and C2x.

Myers: relevant to users of previous editions – this went through corrections rather than the new feature process.

Bhakta: Jim and Fred did a really good job!

5.16 Thomas, C2X proposal - min-max functions [n2532]

Bhakta: This addresses issues IEEE found with existing operations; adds new functions to replace those in two variations to handle NaNs; name is equivalent to operations from 2008, which didn't specify how NaNs were handled; we add *new* functions for the 2020 fixes but leave the old ones in place for compatibility with old code, while also providing the non-problematic functionality.

The binding table removes the 2008 functions; `fmin` and `fmax` are from 1985; we're removing `fminmag` and `fmaxmag` from 2008, hoping there are few existing implementations, but they can stay as extensions. We add functions which handle NaN and +/-0. These were edge cases but the specification has been tightened.

Myers: all of this will apply to TS, to the Annex not yet in the draft?

Bhakta: Yes.

Straw Poll: would the Committee like to add the functions described in n2532 to C2x?

Result: 10-3-6 (clear consensus)

Gustedt: Naming explosion - yes on the features, but no on the paper!

5.17 Thomas, C2X proposal - powr justification, wording [n2491]

Bhakta: Editorial, but wanted to bring suggestions to the committee, and address the question of why this exists. The footnote explains this concern.

Straw Poll: would the Committee like to add the changes described in n2491 into C2x?

Result: 15-0-3 (adopted)

5.18 Thomas, C2X proposal - note about preserving math function properties [n2492]

Bhakta: More recommended practice, not otherwise asking the Standard to change, but trying to get as many useful math properties to apply as possible. IEEE requires correct rounding, though C doesn't do so for everything (reserving `cr`). Properties are not mandated, but want to guide the implementation to do so where possible, to get more valuable reasoning for the user.

Tydeman: The `cr` prefix was changed to `cr_` previously.

Bhakta: The diff needs to include that.

Straw Poll: would the Committee like to add the "recommended practice" note in n2492, modulo changing the prefix `cr` to `cr_`, to C2x?

Result: 13-0-3 (adopted)

5.19 Tydeman, snprintf [n2495]

Tydeman: implementations differ, and take both options for the ambiguous statement. One extra sentence is needed to make it clear that the output is unambiguously null-terminated if the result is not negative.

Bhakta: don't see the original as ambiguous – it looks to mean it should always be null-terminated except when N == 0, which the fix doesn't address, and makes worse because there is nowhere to write the null terminator. Am I wrong?

Tydeman: I found at least one implementation that doesn't.

Bhakta: I consider that an implementation bug. The purpose of this is to always null-terminate.

Sebor: I agree with Rajan, I don't think the new working is correct, though clarifying would be fine. The new wording suggests that it may not be null-terminated when the result is 0; I think the guarantee is already unequivocally in the description, and don't see a need.

Svoboda: I see ambiguity in paragraph 3, which never says null-terminated; paragraph 2 says nothing is written when N == 0, not that it is null-terminated – this is the clear case. In favour of a rewrite to be less ambiguous and like making sure, so favour the new wording.

Gustedt: Agree with David – if people perceive ambiguity, clean up the language. The proposed wording is problematic - “if output is written into the target, the output is null-terminated” would be clearer. The next sentence is about complete output.

Bhakta: OK with Jens's suggestion; surprised by David – the first part is wrong!

Svoboda: the text from Fred contradicts paragraph 2, right – I favour cleaning up the language but paragraph 3 needs more attention. OK if “if positive, the null-terminated”, but not non-negative.

Tydeman: I will work on this some more.

Wiedijk: So, as a reader, I want clarification – are we still required to put a null-terminator even on 0?

Myers: this is ambiguous – must we have a null terminator in case of an error? It only refers to an encoding error, what about out of range of the integer to return?

Tydeman: this is implicit undefined behaviour.

Thursday

5.20 Tydeman, Range errors and math functions [n2506]

Tydeman: ambiguities in the math functions have led implementors to interpret them differently; some treat Inf as a range error, not an exact value. This was a joint effort by the CFP group.

Bhakta: Standardese – what does “too close” mean?

Tydeman: There's no good way to describe this, don't know how to handle the term better.

Ballman: Is “too close” clearer than large or small?

Tydeman: we also add what it's too close *to*. This is an incremental improvement.

Wiedijk: So if it's closer than some number, then there is a range error – should this value be implementation-defined? Or the same under all implementations? Is this determined by IEEE-745, or do implementations have leeway?

Tydeman: It's what IEEE-754 defines as underflow – an inexact value smaller than the smallest normal exact value. `math.h` defines underflow and this.

Bhakta: the difference is based on the floating-point model, on its precision and bases, so if that's not IEEE, it will be different.

Wiedijk: so it depends on the implementation, but the implementation doesn't specify it? Is this a valid direction or not in the requirement?

Bhakta: if we wanted to, we could specify it as implementation-defined. Depends on the function what the value is, hard to specify and read, clearer this way.

Myers: If there are no subnormals, there is no underflow. This is inaccurate for `fmod`, `fdim`, `remainder`, which always define an exact result.

Tydeman: a subnormal result is still a range error, not required to be inexact. Should be represented if it can and is exact; if not, the described error, not an FP exception flag. Is `fdim` incorrect?

Myers: `fdim` is correct for overflow only, a tiny result will still be exact. No range errors for `div` and `rem`.

Tydeman: I agree. Can we vote with corrections?

Keaton: We can vote but need the paper.

Bhakta: Part of the problem is, Joseph is right for IEEE, not necessarily for other formats. We can't force an IEEE-specific change.

Tydeman: You *can* get a range error on the IBM360, which has no subnormals, so the words are correct as-is.

Myers: a range error "occurs" - but it doesn't occur in the IEEE case, and does in others.

Tydeman: so it should change to say "may occur" when underflow is exact and subnormal.

Bhakta: I am not confident voting without actual words, only confident on voting without these three functions.

Straw Poll: would the Committee like to adopt the changes in n2506 into C2x?

Result: 6-0-10 (consensus to add)

Keaton: why so many abstentions?

Krause: not confident enough to check the changes are correct – didn't feel qualified.

Bhakta: wasn't time to validate the list of functions.

(others echo sentiment)

Keaton: consensus to go forward, then.

Tydeman: I will submit two papers, one without these functions and one with the rewrite.

5.21 Gustedt, A Common C/C++ Core Specification rev 2 [n2522]

Gustedt: this is a slide presentation because the "paper" is a complete revision of the Standard. There is a lot more content than will be covered, looking at selected major points only.

(slides were presented, questions held for the end)

Seacord: does `constexpr` correspond to the attributes we saw previously?

Gustedt: not exactly – it's similar to `unsequenced`, but more relaxed. There are subtle differences.

Gilding: making a cultural observation between the groups, the WG21 members in my company that I showed this to didn't like that it also represented even a small amount of work on the C++ side, didn't think it could go anywhere. I was surprised.

Gustedt: this will be discussed in the joint SG; Herb Sutter was similarly unsure – he likes it but isn't confident about wider reception; so far responses from WG21 have been positive; wider community response (Reddit) was very mixed with extreme positive and negative reactions.

Ballman: *thank you*. This is a wonderful incremental improvement, and I hope for an SG to track it. Very excited about `constexpr` - this is a huge win from a security perspective by moving work to compile-time. Concerned about the amount *not* common in the proposal – expected a base specification, as opposed to unifying-invention. I also really like lambdas.

Gustedt: there are fields where both are far apart, so added content to accommodate that. Other parts are much more difficult, e.g. IO, printing numbers – massive problems with the interface in C, such as `intmax_t`; C++ solutions involve a lot we don't have, and can't adapt their IO patterns, so replacements are needed.

Ballman: it's hard to judge where the changes are big, and unclear where we're *already* common and not calling out needed improvements. This returns to the question of omnibus vs. feature papers.

Bhakta: I have the opposite point of view – this is C++-ization of C. What is C's advantage? Why not just use C++, if we're just adding features?

Gustedt: this represents a complete underestimation of C++, which has a *lot* more things than just minor augmentations. Advantages are in the C model of evaluation and conversion, not having references is good for optimization, no complicated copy constructors, PR-values, etc. things that are hard to understand and have huge baggage. Avoid C++'s long compile times.

Everything here exists in some C compiler in some way, so if we do these things in C, let's take them as-is rather than reinvent incompatible solutions.

Bhakta: I agree, but in a functional way, how is C different from a subset of C++? W Hat are its advantages as a language? It seems like this means they're not different.

Gustedt: I don't think that C++ can be reduced to something and only keep the parts you want. I really want to keep the C rules for evaluation.

Seacord: A philosophical point: Jens did work on the principles, but we need to agree on what the principles are for what we want to achieve, which is not clear from the proposal.

I'm opposed to breaking existing C code for example. I don't think that principle is applied here in this proposal.

Myers: I agree as well. See principle 6(c) in the charter, and "let C++ be the "big" and ambitious language" in principle 10.

Ballman: Rajan's question is good. For example, `constexpr` is important to C – it allows more efficient code than I can write today, need const-eval on my new arithmetic type to get efficient table generation; the win was in the const property that there is no UB – the evaluator catches overflows etc. - which means the entire interface can be tested with `static_assert`, and is correct if it compiles! So you can add safety with compile-time tests. This isn't C++-lite – this enables things there's no other way to do.

Krause: I'm worried that all these new features but a large burden on implementations; currently there are hundreds of C compilers, but not even 10 C++ compilers. With too many features added to the language, maintaining a C compiler will no longer be possible for small organizations.

Uecker: I like the features, but have philosophy concerns. We should not try to unify. All for compatibility, but different languages with different purposes. Should think about common uses. I am nervous about keeping the ability to evolve C away from C++, too.

Gustedt: this is not *all* we want for C – for example, `defer` - and that's OK, just that when we do integrate, we should see if C++ has a mechanism we can reuse.

Wiedijk: “Core” sounds like an intersection, but lambdas aren't in C today. Should there be separate work in two branches – conflicts in C today, vs. new features we can backport?

Do lambdas imply heap-allocated storage? Can allocation or free fail?

Gustedt: no, the compiler reserves space on the stack. This is why they can only be returned from `inline` functions, so the capture set is visible to caller. Size depends on captures.

Meneide: I think out of everything in the core, only constant expressions would be the back breaking bit for making a C compiler.

Gilding: Question about lambdas was interesting because there *are* differences there in the existing implementation experience - the way Clang implements its blocks is quite different, and heap-allocation for captures is the price you have to pay in order to be able to put an opaque name on the lambda types, and also to be able to not use `auto`. Stack-allocation is more C-oriented but limits what the lambdas can do. Shows a conflicting assumption about usage.

Bhakta: have we considered the implementation cost? C is usually one-pass, this may not be possible any more; this is a lot of burden to some – C's benefit is that it is implementable both by small companies, and by big companies really well – can't just consider usage.

Gustedt: right, we need to consider each feature, whether to add and how. Not all are complicated, e.g. `auto` should be one-pass – types are already deduced from initializer for e.g. arrays, so deduction as a concept is already there. With `decltype`, you know the type already, and `_Generic` etc. use the same mechanism. Lambdas are more complex and need feature discussion. `constexpr` is complicated in the description and needs real work to check.

Sebor: Impressed, kudos for a huge effort and a huge document. What is the goal from WG14?

Gustedt: to share common knowledge, to get sentiment from WG14 about cutting off rough edges and doing work in common; to get feedback on features people like and want to promote, vs conflict; how to add those features in a way that is compatible with C++.

Sebor: We have this in the Charter? At a high level, we're in favour, but how far do we go? `constexpr` is in headers, but is it gratuitous or novel? The extent is not formulated. Nice overview and ideas but not hearing what you want as a motion.

Gustedt: this! You're asking the right questions, such as the goal in the Charter. What *do* we do, not what are we supposed to do.

Gilding: For me the philosophy of C is explicitness: what you write is what you get, as opposed to C++ with its implicit conversions, constructions, implicitly-inserted operations, out-of-line code generation (templates), and so on. In C there is a 1:1 between code and operations. None of the features proposed here contradict that principle.

Myers: lambdas with `auto` introduce much more complicated type inference. Can't clearly check all constraints until the point of use – this feels like template instantiation. I doubt this feature fits in C.

Gustedt: partially agree – needs discussion. The intent here is that type is deducible at the point an expression appears. We should discuss each feature as it comes up.

Seacord: call to figure out philosophy. Does the Charter work like the Rules of Robotics? What is the priority of the rules? 12 overrides the others; principle 1 is a hard line, I don't think the committee wants this.

C has to evolve and match changes in other languages; in some aspects C++ *is* better than C, we can follow its path and improve C as it did. `constexpr` and lambdas are wonderful.

Gustedt: Thanks. Paper observes the principle of breakage - `register` and `restrict` are not broken, C is kept as it is.

Bhakta: actually you change the character sets – explicit Unicode breaks some implementors.

I have opposite stance from Alex on these features - `auto` and `decltype` are not WYSIWYG. Code is read more than it is written. Java bans this, disagrees on explicitness.

Gustedt: `auto` is not just for known types, it also supports type-generic programming. Tool linters can enforce a style otherwise. The possibility is a gain.

Pygott: I expected a smaller document. Where there is already overlap, do C and C++ mean the same thing? Do we define a syntactic or semantic core? e.g. for enums C and C++ use the same keyword but disagree on semantics.

Gustedt: tried to note where the languages differ. The first goal is to identify the intersection, and the second is to enlarge it.

Pygott: we should extract the shared core with differences and talk to WG21 about it.

Ballman: for bit-fields, have you considered ext-ints?

Gustedt: similar but not the same – we already have fixed-width. This is consistent with what some compilers already do.

Myers: adding `bool` as a keyword does break existing code, lots of programs typedef this.

Gustedt: we already agreed this change.

Bhakta: for individual features, it's important to know what you link. Hiding through type-generic macros is in many places in the document and often mandatory. Hard to link to specific versions.

Gustedt: Users seem to want to evaluate, call, get pointers – trying to crystallize what users want. All the old names remains reserved, just don't want users to use them directly. If they collapse to the same function as before, they can get it. Leeway for the implementation to do other things and add new types easily.

Myers: C defines the ABI boundary for other languages, in object files and runtime FFI lookup. Names must be predictable for other languages, interfaces need to be convenient for them.

Gustedt: only the in-language UI is type-generic, as it is in C++. This doesn't add a burden.

Bhakta: Some of the proposed attributes add semantic changes, but need to be removable, for instance they can change structure sizes. `writethrough` could generate incorrect code if misapplied, like `noreturn`. Is there intent to do anything if used incorrectly?

Gustedt: UB, the variable will not be initialized.

Bhakta: for existing programs, analysis won't see DF attributes, and will produce spurious warnings.

Gustedt: this is a very specific hint to silence a common warning.

Ballman: using the storage class for deduction is incompatible with C++ - is this intentional?

Gustedt: the intent is to introduce with the least work needed in compilers – it's an easy change to the compiler because this is already allowed by the syntax and forbidden by constraints. We only need to add a rule to allow `auto` alongside other storage class specifiers. This follows the current syntax and doesn't require any more lookahead.

Myers: what about the `char`/string type incompatibilities?

Gustedt: I will add it.

Bhakta: this explicitly leaves out the floating point parts, but the voted-in floating point content isn't complex?

Gustedt: two aspects – the interface adds thousands of identifiers, so it is complex. Decimals aren't in C++ yet, so they don't fit in a common Core.

Bhakta: I don't want this to drop parts of the Standard that already exist.

Keaton: implementation experience from Chapel last year was that type inference was the largest, most complex, most expensive part of the compiler. The code is hard to read and hard to trace-through when debugging.

Gustedt: can't debuggers help with that?

Ballman: reviewing code with `auto` can be difficult in C++. Many guidelines forbid it.

Gustedt: again, I mainly want this for type-generic code that *needs* it – almost-always-`auto` is a style question.

Gilding: In C++ I use `auto`, but it solves a different problem with noisy long template type names that don't communicate useful or readable information. C doesn't have this problem, it mostly has short type names and fewer types overall, so users will need it less.

Wiedijk: I use OCaml, which infers everything. Reading the code is easy, but errors can be incomprehensible.

Seacord: we're not producing votes, should this be relegated to the new SG?

Gustedt: I would like feedback; if we create the new SG, this will not likely be the basis of discussion.

Meneide: I have personally found `constexpr` extremely helpful, I have to use macros or implementation-specific behaviour, not even Clang and GCC agree on what is constant. Constant expressions need improvement to reduce macro use and remove the need for platform tricks.

Inference for `decltype` is helpful, can be used to implement things like a safe `printf`. `auto` might be less helpful, unwieldy type names aren't a C problem. Without lambdas there's no driver for it. With lambdas, we need `auto` to support stack-allocated lambdas so the type can be named. I am wary of `auto` parameters – useful but need a framework to ensure they stay in the spirit of C.

We can get a lot of this into C and preserve its usefulness while tackling the issues people use extensions to solve.

Gustedt: this is the first time much of this is proposed – I am prodding for a reaction.

Bhakta: I *don't* want `auto`, lambdas, `constexpr` - I care about the whole market, including small compilers for small systems. C lets you write a one-pass compiler you can understand. Semantic differences are useful, but new features should be useful for C, not justified by being in C++. We should clarify extant features and propose new features separately.

Pygott: agreed, we should focus on what we already do.

Gilding: if users are currently relying on vendor-specific extensions to provide features that already exist in C++, that provides a compounding justification for standardizing the existing practice, as C users demonstrably already need it. We should review what extensions are in wide use.

Ojeda: `constexpr` is almost universally useful but hard to see if all compilers can implement it. On lots of compilers, nothing is implemented by everyone.

Ballman: Are we saying there is implementation burden for technical reasons, or just the scale of effort proposed? We never required C to be implementable by one person – where do we draw the line? C will fall behind, this is not a fair restriction.

Myers: common extensions were listed in the build-up to C11, n1229. Some were not added, but we could do it again.

Bhakta: I didn't imply compilers should be one-person, there are extremes to the tradeoff, but some features like `constexpr` are a more significant burden than e.g. enums. There's no rule but it is important to consider. In the past, one-pass was considered a reasonable goal. "Reasonable" needs to be flexible for the needs of different targets.

Gustedt: Thank you all for your time and feedback.

Keaton: JeanHeyd Meneide has requested that we skip 5.24 (n2500), so the slot will go to 7.2 (n2528).

Friday

5.22 Uecker, Free Positioning of Labels Inside Compound Statements [n2508]

Uecker: this fixes a minor annoyance in the language which is hard for beginners to grasp. Can also be annoying when `#ifdef` removes code in certain configurations. Not a problem for attributes, since they appertain to the label. Label is added at the same level as other block items, implicit null statement means it works as expected in front of declarations, control flow of valid code is unaffected. Adding to the grammar allows us to remove noisy examples and discuss directly.

Gustedt: this is a perfect candidate to fist discuss with the C++ SG.

Uecker: will check, but think C++ already has this.

Gustedt: what happens to initializers following the statement? Should we vote that initializers are also unaffected?

Uecker: No, there is always an implicit null statement inserted as the "next statement", so nothing will be skipped.

Ballman: this introduces an incompatibility: label followed by close brace is not permitted in C++. Not opposed but should put to WG21. I am willing to help with the C++ proposal.

Gilding: does this introduce a risk of user confusion? Say the label is written as though it is the only element in an unbraced-for construct, and there is following code.

Uecker: I would consider just removing labels outside braces, find it confusing. Could add an additional change to obsolete this which would affect macros generating control statements.

Myers: The syntax for *labelled-statement* repeats three times, it should only be present once.

Gustedt: Oppose the additional change – this feature is used a *lot* in macros, for example the `defer` library implementation. This would break macro extension of the language. Not good for usual programming, but useful.

Straw Poll: would the Committee like to add n2508, subject to deleting duplicate rules from the grammar, into C2x?

Result: 13-1-2

Straw Poll: would the Committee like to add the optional change in n2508 into C2x?

Result: 2-5-7

5.23 Uecker, Compatibility of Pointers to Arrays with Qualifiers [n2497]

Uecker: this is an old issue, last seen in April 2015; not a defect but a potential addition; consensus was favourable so brought forward as a feature.

This already works in C++ ; commonly-implemented extension. The wording is taken from C++, and only affects CV-qualifiers, not atomics. This has no bad effects because qualifiers only work on l-value conversion, but arrays decay before use, so it has no direct semantic effect on them, but the type does change – arrays now get CVR-qualifiers on their pointers.

Myers: editorial – footnote should be “this rule”. Arrays with elements also applies to multidimensional arrays, but the current wording may not allow this? The element type is recursive and both array types gain the qualifier. “Arrays whose element” doesn't cover this, needs a wording change.

Wiedijk: can editors fix typos and so on silently, or does it need to go through committee?

Keaton: editors can fix *genuine* typos, but check with the author.

Ballman: this is Creative Commons licensed – does that need removal?

Uecker: ISO has an independent licence to every submission, but this gives other users the right to use it independently.

Keaton: we prefer not to think about it, but ISO owns all n-documents.

Straw Poll: would the Committee like to adopt something along the lines of n2497 into C2x?

Result: 14-1-2

5.25 Svoboda, Change request for C17 s3.4.3p3 [n2517]

Svoboda: this is a bug report on wording, not semantics, though it came from integer safety group.

Integer overflow is only defined for signed – the Standard is inconsistent, doesn't define it. Defining it would be a lot of work, and replacing the example is simpler.

Gustedt: Favour this, though an even simpler change would be to just add `signed`.

Svoboda: students are often unaware of this issue – dereferencing `NULL` is clearer.

Keaton: the distinction isn't needed because unsigned is defined not to overflow.

Wiedijk: Agree to both – putting `signed` in would solve the problem, but `NULL` is a better example for pedagogical purposes. Lots of UB is not "obviously" undefined, and this is an example.

Gilding: Strongly agree that this is a much better example because it is immediately intuitive. The overflow example sends the user on an unrelated tangent thinking about whether it's really undefined and when, which is not the point here.

Straw Poll: would the Committee like to add the changes in n2517 into C2x?

Result: 17-0-1

Ballman: this is an expensive way to change examples! There should be an easier process.

Keaton: examples are special, even if the change is editorial.

Ballman: as part of an editorial omnibus paper this might make more sense.

Gustedt: maybe running through the editor group first would help.

Bhakta: I like papers for something like this that changes the whole example. Test suites and documentation copy examples wholesale, so I consider this a big change. I wouldn't want a paper for the `signed`-only change.

Wiedijk: so we could make a small change editorially? There's a spectrum of changes with this on one end, and Core on the other that is too big to ever pass; how do we address this properly?

Keaton: we'll add an agenda item for the next meeting.

Meneide: so if someone asked to add a word or fix a typo, we'd just add it. If you bring a change that touches the semantics, editors will redirect it. Papers have typos all the time, we check with the author but it should be "taken care of".

Svoboda: I assumed a Clarification Request would be a lighter process.

Keaton: this is a Change Request – Clarification Requests are questions.

7.2 Stoughton, Removal of deprecated functions [n2528]

Stoughton: The Austin Group tracks C and tries to defer to C on overlaps, but noticed that obsoleted functions from POSIX had been added to C2x and need direction – should these be maintained? Or was C following POSIX by mistake?

Gustedt: I did that, didn't notice the deprecation. Good that the Austin Group is tracking C. We are missing an Agenda item for liaising with the Austin Group. Would have proposed these anyway, as they are *not* locale-dependent. `strftime` drags in a lot of code, while `asctime_r` can depend on no other library calls.

Myers: POSIX has explicit locale-argument versions, and has also obsoleted the non-`_r` versions of the functions. Should we remove those too? If not, we should keep the `_r` functions.

Seacord: maybe we should follow POSIX's lead.

Gustedt: either follow and remove all functions, or offer users the thread-safe versions – all or nothing.

Stoughton: I am only seeking direction to report to the Austin Group.

Bhakta: I objected to adding these for other reasons, but obsolescence is a strong argument to remove. `strftime` exists based on use, so as a replacement it can be negative or positive depending on what users want – the userbase uses locales a lot.

Gustedt: `strftime` does many things – these are tiny functions that don't need locales for the functionality they provide.

Stoughton: Not being in the Standard doesn't mean a vendor can't provide them.

Gustedt: Libraries are only allowed to use reserved names.

Keaton: do we need a new paper? It may have to make other changes.

Stoughton: I would write that.

Straw Poll: is the Committee generally in favour of removing `asctime_r` and `ctime_r` from C2x?

Result: 8-3-7 (sentiment to see a real paper)

Bhakta: I would like to see more feedback from POSIX on this kind of thing – it's helpful to have before we vote in content.

Tydeman: does the Austin Group care about the thousands of new math functions?

Stoughton: They exist by reference.

Myers: POSIX had signalling-NaN long before C – will it update for consistency?

Stoughton: We can try. Let me know abut other areas of concern, or join the Austin Group calls.

7.1 Krause, use const for data from the library that shall not be modified [n2526]

Krause: this hasn't been done yet for compatibility with ancient code, similar to the missing `const` qualification of string literals. Implementations can warn anyway, but this makes it easier to diagnose bugs. The change can break existing correct programs, but I couldn't find any real examples.

Svoboda: A similar change to `getenv` was proposed earlier and rejected on compatibility grounds. It would not break any existing code that actually worked.

Wiedijk: I sue C for quick'n'dirty programs and don't use `const`. Do I now get a compile error?

Krause: Yes. And if you don't understand `const`, C is not the language for you!

Myers: this seems close to n2417, similar to `time.h` - what was concluded there?

Krause: I don't know.

Ballman: why doesn't this proposal mention the `const`-losing functions?

Krause: those have a legitimate non-`const` use case, to find something to modify. That's a bigger change than just the signature of a return, and would break many safe uses.

Gustedt: I favour any change improving `const`-contracts! This is the right direction. In London in 2016, we proposed type-generic string functions, which had no implementation experience, without providing headers. We can still do this for C2x.

Gilding: `const`-losing functions are addressed in n2522, but that's a big conceptual difference from library QoI. Qualifier-generic functions that preserve argument contracts are a much broader concept.

Sebor: there is no constraint preventing this as-is, so it's unsafe. I agree that there's no ABI impact.

Myers: the constraint is described under “simple assignment”.

Krause: users who really need this can always just add an explicit cast, if the code is too complex to propagate the qualification.

Straw Poll: would the Committee like to add the changes in n2526 into C2x?

Result: 14-2-3

Krause: everyone who spoke was in favour – why the No votes?

Bhakta: this breaks existing code. The committee is too open to breaking programs.

Stoughton: this will break thousands of programs. POSIX never change existing practice if they can.

Wiedijk: “breaking”? GCC only gives a warning. The code is the same, so where is the breakage?

Stoughton: warnings indicate something the Standard doesn't permit, and compiling without warnings is a requirement for many programs.

Wiedijk: What about just adding the cast?

Stoughton: that means the code has to change.

Bhakta: many clients don't allow compiler warnings. This would be a breaking change. The committee is happy to say we can break things that are “easy to fix”. Users should be able to use new features without breaking the old ones. We can't just tell users to compile in an older Standard mode when ISO doesn't acknowledge that older versions of the language still exist. This is not fair to users.

Svoboda: modifying this was already UB, so adding `const` doesn't change the meaning of a valid program. The breakage is of code that should break.

Bhakta: I agree on the semantics, but it means a new warning on a codebase that was clean.

Keaton: does anyone's vote change as a result of this discussion?

Sebor: If we want to indefinitely support existing code, we are very constrained. I don't support compiling sloppy code forever with no diagnostics. New versions of compilers always add warnings, and people deal with it – we can't constrain ourselves from safety and usability changes. It's an improvement, and legacy code may have to change in order to migrate.

Ballman: If the Austin Group had a veto, would they use it here? This would change my vote.

Stoughton: Yes, if we had one. This will break the POSIX Charter.

Wiedijk: so if we cast off `const` but don't modify it, that's not UB?

Krause: not if it's only read.

Myers: if you had a veto – for only some functions, or for all? Some will be more breaking than others, e.g. `strerror`.

Stoughton: `getenv` is the worst. All change a definition.

Keaton: we should reconsider this for a liaison issue.

Seacord: I propose we defer this to the next meeting, where Nick can elaborate on the problems.

Gustedt: I disagree, we should choose our opinion and send it to POSIX, and get their response.

If we undo it we should have a second vote.

Ballman: I agree with Robert, want to see more – but we voted on it, and it risks falling through the cracks. We need a note.

Keaton: this really ought to be deferred to the next meeting.

Straw Poll: would the Committee like to defer action on n2526 until the next meeting, in anticipation of hearing from POSIX?

Result: 12-1-3 (deferred)

7.3 Pygott, Allowing the programmer to define the type to be used to represent an enum [n2533]

Pygott: this was discussed at the London 2016 meeting and accepted in principle. We take from C++ the ability to specify the representation type. Terminology is taken from C++.

Nothing changes the semantics of old-style enums, so no existing code breaks. There are new potential constraint errors on value ranges; incrementing follows "normal" arithmetic. We also add the C++ constraint against implicit conversion.

What other types than integers should be allowed to be specified?

Ballman: C++ allows `bool`, but implementations get this wrong in practice.

Gilding: This could surprise users by changing the type used in a library interface if the library is upgraded, so clients of upgraded libraries could be affected. Not objecting.

Gustedt: this finally allows forward declaration of enums, because we know their size. I dislike the weakening of expressiveness by removing explicit conversions. Use as e.g. flags is important. Casts are user-unfriendly here.

Pygott: but if we change the syntax to match C++, should we take the semantics too?

Gustedt: there is no ABI incompatibility.

Bhakta: it wouldn't break compatibility, as it only removes a restriction. C++ code will work in C. The change to disallow wraparound in the definition differs from initialization elsewhere in the language – what is the rationale?

Pygott: MISRA rules require constants to always be within range, but we could remove it.

Wiedijk: why do we discard CV-qualification instead of forbid it?

Pygott: this allows for typedefs which include qualifiers to be used.

Ballman: in C++ a narrowing conversion wouldn't be allowed – I oppose removing that. I find the type safety argument very compelling, letting the type system prevent use of the wrong enum improves the safety of the language.

Gilding: In C++ users of strongly-typed enums end up overloading the bitwise operators a lot of the time anyway, because C++ users also hate writing the casts. I support the type safety improvement but QoI means this isn't necessarily a blocker. C++ sets expected constraints anyway.

Stoughton: is it clear what the type is? Can it be bigger, e.g. 128-bit?

Pygott: It's specified as an integer type, which right now allows enums, `bool`, and ext-ints.

Krause: the silent discard of qualifiers is surprising. Disallowing would be clearer, even if it is QoI. An implementation would warn about the value range anyway, but I would rather have it. Needing the cast is ugly, and might discourage user adoption, but improves compatibility with C++. DO you want to allow other enums as the base type?

Pygott: inclined against because it risks circular declaration.

Gustedt: range requirement is a good idea especially for unsigned. Classical enums have UB if they exceed `INT_MAX` - this is now defined or constrained. What would `bool` mean? There are problems with the conversion, allowing this is a trap. For conversions, I agree on security; we could mitigate it by allowing enums in operations so that if the operands are enum, the result is too, providing safety and convenience.

Bhakta: I don't understand Aaron's complaint, just write with the cast when writing code for both languages.

Ballman: If I write a header developing in C, I might forget the casts.

Gilding: allowing enums as the underlying type would be intuitive to define families of related enumerations. Would disallow `bool` if we know compilers get this wrong anyway. I support typed overloads of the operators for enums.

Ojeda: Do we want to allow some way to query the underlying type? C++ has this.

Krause: enums can represent two states other than `true` and `false`, so `bool` is a natural choice for these. It's an integer type, just one with padding. This is consistent with user expectations.

Pygott: `bool` has associated behaviours, e.g. testing 0 or non-0 – what happens with a third member? How are they associated with `true` and `false`?

Krause: the same as giving anything more members...

Hoeffner: With operators, does this provide a new way of evaluating the result, or are we disabling promotions? It's confusing to only use the size property.

Tydeman: are `e1` and `e2` the same type or different for the purposes of `_Generic`? What type do you get?

Gustedt: right now, it goes off compatible type. For `bool` - as data, it's an integer, but not one for conversions. Does the conversion do the same thing as the underlying type? This is confusing if the underlying type is `bool`, so I oppose.

Ballman: using an underlying enum is not allowed in C++ because it may in future be used for extension via inheritance – we shouldn't close off the design space for C++.

Pygott: For the member values, should out-of-range be a constraint or a conversion? Is the need to cast from `int` desirable or not?

Seacord: inconsistency is confusing.

Straw Poll: should implicit casts from int to new enum types be permitted?

Result: 4-7-8 (no consensus)

6. Deferred Papers

7.4 Blower, Adding Fundamental Type for N-bit Integers [n2534]

7. Resolutions and decisions reached

7.1 Review of decisions reached

Straw poll: would the Committee like to see something along the lines of n2539 in C2x?

Result: 15-0-1

Straw poll: does the Committee wish to adopt n2481 into C2x as-is?

Result: 16-1-1

Straw poll: would the Committee like to adopt n2527 as-is except for the section titled "Allow attributes on an expression in the clause-1 position of a for loop" into C2x?

Result: 17-0-1

Straw poll: would the Committee like to adopt something along the lines of the for-loop proposal from n2527 into C2x?

Result: 12-1-4

Straw poll: does the group want to see progress in the direction described by n2493 to include informatively-reserved identifiers into C2x?

Result: 12-1-2

Straw poll: does the Committee want something along the lines of n2484 casting added to C2x?

Result: 6-3-6 (no consensus)

Straw poll: does the Committee want something along the lines of the memory reuse described in n2537 to be added to C2x?

Result: 9-1-6 (direction yes)

Straw poll: would the Committee like to see a non-elidable, non-optional memory-erasing function added to C2x?

Result: 14-0-2 (clear direction)

Straw poll: would the Committee like the non-elidable, non-optional memory-erasing function not to specify a value in its interface?

Result: 6-5-6 (unclear direction)

Straw poll: would the Committee like to be able to specify a value in the interface to the non-elidable, non-optional memory-erasing function?

Result: 7-4-6 (clearer)

Straw poll: would the Committee like to have both no-value and value-specifying interfaces to the non-elidable, non-optional function available?

Result: 5-6-7

Straw poll: would the Committee like to see something to specify that the address of a global cannot be taken?

Result: 8-0-9 (clear direction)

Straw poll: would the Committee like to use the keyword `register` to prevent taking the address of a global?

Result: 8-6-3 (no consensus)

Straw poll: would the Committee like to see an attribute that would mark an object as non-addressable?

Result: 12-5-1

Straw poll: would the Committee like to allow access to non-thread-local const objects from signal handlers, along the lines of the proposal in n2523?

Result: 16-0-3 (clear direction)

Straw poll: would the Committee like to allow signal handlers to refer to objects without accessing them, unless they are thread-local?

Result: 16-0-3 (clear direction)

Straw poll: would the Committee like to make string.h mandatory for free-standing implementations, minus the functions `strdup`, `strndup`, `strcoll`, `strxfrm`, `strerror`?

Result: 12-0-6 (clear direction)

Straw poll: would the Committee like to see further proposals to make more of the Standard Library mandatory for freestanding implementations?

Result: 13-1-5

Straw poll: does the Committee want to remove the new `intmax_t` functions?

Result: 6-7-6 (no consensus to remove)

Straw poll: does the Committee want to remove the old `intmax_t` functions?

Result: 5-5-9

Straw poll: does the Committee want to remove the `%j` specifier from the `printf`/`scanf` family of functions?
Result: 6-9-4 (no consensus to remove)

Straw poll: would the Committee like to add the functions described in n2532 to C2x?
Result: 10-3-6 (clear consensus)

Straw poll: would the Committee like to add the changes described in n2491 into C2x?
Result: 15-0-3 (adopted)

Straw poll: would the Committee like to add the "recommended practice" note in n2492, modulo changing the prefix `cr` to `cr_`, to C2x?
Result: 13-0-3 (adopted)

Straw poll: would the Committee like to adopt the changes in n2506 into C2x?
Result: 6-0-10 (consensus to add)

Straw poll: would the Committee like to add n2508, subject to deleting duplicate rules from the grammar, into C2x?
Result: 13-1-2

Straw poll: would the Committee like to add the optional change in n2508 into C2x?
Result: 2-5-7 (did not pass)

Straw poll: would the Committee like to adopt something along the lines of n2497 into C2x?
Result: 14-1-2

Straw poll: would the Committee like to add the changes in n2517 into C2x?
Result: 17-0-1

Straw poll: is the Committee generally in favour of removing `asctime_r` and `ctime_r` from C2x?
Result: 8-3-7 (sentiment to see a real paper)

Straw poll: would the Committee like to add the changes in n2526 into C2x?
Result: 14-2-3

Straw poll: would the Committee like to defer action on n2526 until the next meeting, in anticipation of hearing from POSIX?

Result: 12-1-3 (deferred)

Straw Poll: should implicit casts from int to new enum types be permitted?

Result: 4-7-8 (no consensus)

7.2 Review of Action Items

Keaton: To coordinate with Herb Sutter on establishing co-located study groups between WG14 and WG21.

Stoughton: To submit a proposal on the removal of the removal of `asctime_r` and `ctime_r`.

Stoughton: To write a POSIX response to n2526 before the next committee meeting.

8. PL22.11 Business

Placeholder – no official PL22.11 meeting was held.

9. Thanks to host

Thanks and apologies to Philipp Krause, the originally intended host.

Thanks to ISO for supplying Zoom capabilities.

10. Adjournment

PL22.11 motion by Bhakta, Tydeman. Approved.

Adjourned.