69th meeting, 'nice'

WG14 2024-01-22 INRIA F2F!

Thanks to Jens,

US, Can, Aus, UK, Fr, Swe, Pol, (NL)

Hybrid rules require meeting stops if network stops
figure more going fwd
straw polls not real polls   RB: can we establish what is consensus?
↳ no draft to vote stuff into anyway rn, so no final changes; not too important

1.4 CoC & guidance | RB has a voting program (written in C!)
shown, IEC shown not discussed

1.5 Approval of prev minutes   AC / JL / No obj

1.6 Actions & resolutions - RCS to submit - held/deferred to review
1.7 Approval of agenda N3213   RB/DP/No obj
1.8 (above)
1.9 schedule - DIS comments are back since Dec 11th
must process the comments, only mandatory thing
options: tech changes → FDIS
ed. changes → no more steps, more time to publish
auto-cancellation on 12-Jul, will not use it
C2Y papers if we finish DIS

2 Liaison
2.1 ISO, JTC1, - none
2.2 NBs - PL.C meeting w/ ballot items on 5th
2.3 WG21 - RCS has attended, typeof voted down wff, upset JHM
'we should do our jobs even if they don't'
(AB: any time someone mutes, we lose audio (Jens's device)
↳ will try, at least until change good

C++ added decltype instead; C adds tlt_uq; C++ hate this

more votes for than against but not adequate consensus; co-chair Nino
is quite active
? CB: not compat already
RCS more import to C++ → header compat is critical
MU nothing comes from the C++ side → Atomic? (did do this)
RB request AB's view → AB couldn't lead, still attends

2.4 WG23 - CP has resigned, no one else
2.5 MISRA C - new doc etc
please use 'C23' not '24'
2.6 Austin - nobody
2.7 Unicode - nobody
2.8 no other groups

3 Study Groups
3.1 CFP working on TS pt 4+5, not much time bef. expiry
issues for C2Y, welcome issue submissions
to vote this meeting, submit unless advised not
3.2 C MoM → push TS 6010 fwd or not
(discuss now) ISO disapproved Jens's changes, unsure direction
prev. decided to rewrite per ISO req., but relitigated
do we want to integrate now or publish the TS?
RB: thanks Henry! prefer TS - more likely for implementers to
examine it sooner, don't jump gun
AB: prefer TS; at Intel, changes sched. a year in advance, nobody
starting in 2024 even, need chance to investigate
very bad mark if goes into IS & not implemented
very high effort to roll into IS & remove again, high burden
AC not opposed options to publish now and vote into IS (not worse)
Jens no substantive changes in three years & voted pos. by all NBs
do want to know sentiment of room; would be work to rebase
on draft, then vote, need direction to not waste time
RB sim. to AB, same issue at IBM, same timeline; do not want in
IS directly; IBM need a pub. doc not N-doc

the CFP uses TS for this purpose, experimental changes; many changes already came from experience

CJ need impl exp - unclear why need a TS for this; want to avoid ISO overhead & put to vendors to do experiments; how can TS make a stronger case than N-doc?

CB very long doc, 50% changes - unclear which choices made? → stated in the doc, one model is proposed concretely

Esk like very much, reflects understanding; worry abt via int; need to see how plays out; 'real' impls are a lot more work though not now to put in IS but it is good

MU not new feature - clarifies the existing model. any non-opt tool is already conforming; opts are diverging

RB IBM won't impl sth not formally published. can do draft stuff but if changes. opt. scenario is what we need to guide the perf team.

Jens doc has definite choice: diff is concrete, compromise chosen by WG14. compilers don't know yet whether conforming; GCC & Clang community have been working on & finding issues
Feel like document resulting process of TS would be unusable as a doc; do sth different from before. ISO direction is bad. and it would be paywalled

AB TS are not a good ship vehicle but have been generating feedback from Clang team - unsure. Not yet sure it's impl-able
best model we can offer but still don't know if it works
could sell management an N-doc but corps want publications & things to plan around
↳ changes were only formatting but might don't know that need to understand process
have been working on it but not per plan

JM not concerned by TS but want experience - consider consume memord as cautionary tale

Esk apply to C23? diff?

Opinion cont. w/ TS 6010? 12 +5 / 0 0 / 3 +1 (yes)
approval 17 0 4

Opinion   WG14 want the TS content in C2Y? ~~19~~ 20 / 0 /3 (yes)

JM must say if exp. exists in vote
RB pointless

3.3 Compat group - no activity r/n, soon to discuss provenance on Feb 1
RB - what do C++ think of it? Jens - don't know. diff't model
MW - want a similar model, diff't foundation

3.4 UBSG - still meeting, published examples doc in October
do want feedback on it

3.5 New SG - no prog. for time being, wait on Community
3.6 info - org as an SG, address various problems
need JHM's opinion, come back to this
AB - need to know if we're using C-docs

4 Future meetings
June - not needed if we don't FDIS,
will know by Friday
'you're smart & pretty, i like taking'
30 Sep in MN
2025 options open; Graz; London; Spain; RCS could arrange
a Jp location

5 Doc review - DIS ballot comments

N3191 most changes are made but can be reverted
using UC for most comments

006 - CP objects to UC; withdrawn by peer pressure

Decision   9 / 1 / 6
+2 11 / +1 / +4 (yes)

'no one fucking cares' - RCS   2   10

CB don't we have to accept ISO cmts? → we can still say no, idle threat
accept the stupid, limit pushback to important, don't disrespect

007 'wait for editor' → changed to 'Program semantics'
↳ ISO editors got *their* numbers wrong

010 rejected by editor (LC also to reject)
↳ semantic meaning, *normative* meaning to bolding
(rationale must make sense to ISO)
011 rejected, normative content change
012 profoundly stupid rejection
013 process conflict

017 - partial accept b/c most *are* used normatively
JM someone to check ISO 80000-3 usage → JM checked, not used
CJ C++ includes it all by Ax
↳ ed believes all except UA31 norm

'shall be' part is kept
JM: whatever ISO wrote is not a Std, garbled
AB: call this 'accept w/ comment'

018 ITTF used to be implicit b/c ISO 10646
↳ still on linked site JM: should cite the IS itself, not location of download
DB: not new addition... old
CB disputes meaning of question
021 similarly
022 rejected, breaks meaning
023 as well
024 term ended up being unused but will be used if GB-005 is applied. 3.9 widely used, 3.10 maybe intended

025 apparent rules conflict
JM- ISO assumes Stds are 20-page pamphlets & moving to CJ makes sense, doesn't work for proglang

according to AB there is an existing exemption - ask Keaton?
Jens had this comment before & just ignored it

027 AB disagrees w/ editor - this is asking for 80% of refs in
the document to change
RCS interprets as only one listed ref
↳ ref outwards from Clause 3, but there are others (JM)
↳ misunderstood - only Cl 3
DB is the issue w/ outrefs to italic ToA outside Cl3?
Ori - choice btw narrow accept & pushback ; RB they do say others too
RCS - breaks their document w/ glossary links ; removing them is still
usable, can still search ↓ in vocab document
FW - these are only non-norm NOTEs
DB - usability merit for this doc ; RCS - decreases it for vocab doc
very useful document of Warren
↳ no ontology, not consistent btw Stds
RB suggests poll ; FW cannot remove a NOTE that makes an entry
meaningless 2+1 7+1 5+3
3 / 8 / 8 (Abs)

Decision
JHM this is probably a reject of comment anyway
they will demand the others too
RCS they won't do that
settling on a qualified reject
RCS 'we carefully cons. our dec. to tell you to fuck off'

030 what does progressing a ToA to remand look like?
031 pulled def to new entry
032 a constraint is a requirement, but the description is not
meta-requirement
033 using CFP words
034 - different kind refs, contains. vs target of reference
neither vn. of wording is very good
Jens - do we even need this term? → could go

sc22wg14 1

039 rejected by UC, though interest
046 CFP was asked about, ~~we should be~~ [24428] refl. provides wording
061 diverging opn. b/w JM & AB - notes not numbered throughout, still have paragraph though
062 'nice'
069 'nice' ; does stating the existence of a requirement elsewhere constitute a requirement in itself
- just avoid using 'ISO normative words' in rewrite
087 ISO have many comments about examples but these are supposed to clarify the normative text - need to check that examples don't then mislead
RCS - would like to accept & leave to editor
AB - 'must be' in 6.7.2.2 → could be 'is' but is meaning lost?
JM - happy w/ 'is', nothing breaks w/ this
CB - it should be possible to understand all reqs w/o reading examples at all
MU - we do that JM - frequent review comment
DB - this is a blunt tool, 'it' refers to an object extenal
Jens - removes part of the content talking about a CVio considering program may not be correct
JM - no good verbiages for consequential examples, applying the reqs leading to conseq. - Directives do not allow an easy way to express this.
RCS - not useful to this meeting. leave to editor
CB - please cite the matching requirement in normative text
RB - bring this one back at the end to review changes
AB + RCB to join ed. cmte

093 - JM accept the tortuous sentence
can't use 'may' and 'may not' in notes
CB - don't directly contradict ISO values

107 - Aushia recommends reject to avoid delay Std
RCS agrees

only to consider for case of FDIS

125 - JC: they want to change 'may' to 'can' but this seems
contradictory RCS: 'might' not 'may'
AB: prefer case-by-case basis
to review
more reply-smithing

126 - MU - this is GB51
6.1 '8' is meant to refer to this

[IBM saves the microphone] ○ multiple mics for MN
meeting

127 - Me - this shouldn't go under A.1
RB: footnote? RCS: not normative? JM: even more subcs
going with new clause, no NOTE

129 - JM: demonstrates why we need stable tags like C++
JHM: working on it, didn't work out for '23
Jens: can we count from zero? → unknown
CB: agree w/ RCS just delete the sentence
DB: renumbering AxJ is not acceptable

130: JM - ref is present, need a 'shall'
131: footnote 9 to become RP
footnote 1 to be main text

~~○~~ ATTENDANE LIST

Tuesday document number for cmt disp, can vote on it

135 may need editor?
138 directly contradicts our convention (refers to O11)
Jens: ISO don't want us having definitions in Clauses, hence this
139 RCS: ISO changed rules RB: yes

RB: still out to ballot pending approval

Jens: mention wide usage of para nums JM: JTC1 res 10 request?

RCS: paste when we find it

JM: NSB01 SC22 general request, resolution 23-05, to JTC1 for p-num at all stages

140 licensing issue (currently Potolino)

145 'most' vs 'none' RB - can we just strike sentence?
JM - editorial recent add anyway
↳ doing so

156 missing ref. in body text

'resting objection face'

157 - Jens: against this
JM: ISO not understanding how users interact, but they don't need our help
RB: add to biblio? → needs to be cited
RCS: removing this doesn't affect page count
RB: refer to having sample impls, cite?
RCS: just call their bluff.
↳ now worrying this implies missing information
AB missing ref in p5 to 'the C Reference Manual'
(this is part of KnR 1 for publication purposes)
CB agree w/ ISO, simple to comply & urls break; implies promise abt content
Me agree w/ removal

Opinion

Would WG14 like sec 180 cut 157, by removing Intro p7
7+1 / 3+1 / 5+1 (Yes)
9 4 6

158 - referring to 'future revisions'
159 - needs to be consistent w/ other naming, type chosen

162 - historical misuse of 'type' to mean 'return type', this one is wrong

Overalignment permit in MB-24?

165: RCS why does this affect formatting?
Sens: this is an automated list
JHM: scripts have been fixed
178: SM - should also move 'perform a trap'
DB should either be alphabetized or organized
RCS, RB agree ; RCB 'value' is fine as standalone
RCS - just flatten everything
SM - two different meanings of 'character'   Me: hard to read as-is

Opinion   Flatten Clause 3?   /   /   (Yes)

SM: would like to lift the three non'value' terms from 3.20
3.20.2 3.20.4, 3.20.5
RB: don't think minimal change is enough ; RCS, CB nervous abt it
FW: these are not values, rep.

Opinion   lift those?   9+3 / 3+0 / 4+1 (Yes)
12   3   5

179 'fucktards' - RCS
181 JHM - only a couple of uses, easy audit
AB - almost entirely non normative
182 strange comment
does actually ignore an existing guideline
CB doing nothing is safe ; our company rules say US sp.
RB make ISO happy
Me reject British-ism
FW don't care, won't affect culture
Sens we should be consistent w/ doc
CB US. vs RoW!
AB WG21 already did this & rejected comment
CP- compat w/ 21
PR so ... Oxford comma? didn't like that
SM Directives say to be consistent, then suggest dicts & Webster as an acceptable option

RCS could be weird to be diff't from '21
DB vote on language for doc. for C2Y Charter instead
FW do mention Webster dict.

186 needs review. not a good change for example snippets
good fix for footnotes & notes
unclear what improvements e.g. init 'below' text
usually advised against for print docs
RCS ↗ → 'remainder of subclause'
CB problematic in footnotes, likely to be rearranged; not all places really can be referenced easily, e.g. printf flags # refers to B
Me ISO told us not to use p-nums
RB - on 185, do want to see change

188 in plaintext we sometimes use commas, not '
spaces could break it (JHM)
JM ref. to O12 not applicable to prog langs
CB too fixed on code vs. prose?
RCS well in C we use a tickmark!
FW think spaces in text are fine
Jens use commas in my book
CB house style issue - 6.4.4.2 could be broken by commas
Me more usable not to have a sep here, for copypasta technical value even in prose
DB/JHM ideally want non-copyable spaces
not worth it at this time; maybe dig-seps in code
PK comma is very unclear by locale; spaces are more readable
JHM don't think we use commas f/n
rejected b/c worried abt. code; do-able for prose
RB agree w/ Alex, this could affect meaningful numbers like env. limits
DB should be doable to be searchable

CB want a straw poll

Opinion   just reject it   13+3 / 1 / 3 (Yes)

189   ISO's number is wrong!

190 is ridiculous, affects normative text
CB   suggest we be diplomatic about avoiding it in future
RCS   give good reason - AB length
Ori   stage of process
Gsk   more reasoning in the reply is good
Me   35 years of precedent
RB   'accept' by moving two or three
↳ no, will ask for all of them to be done
DB   difference?
JM   NOTE is for understanding, footnotes are for context not expl.
↳ probably we aren't consistent on this
JHM   think they want the precedent to end!
CB   emph. that diplomacy is impt here

Decision   accept doc N3216   16+3/ 1 / 0 (yes)

DIS Resolved

Jens, RB, JM, AB, HK   volunteering for editorial review group

3.6 Infrastructure SG (was deferred)

JHM   doc sys not finished b/c editing work. Expect to be working for June; no more editing
paused work will resume; avoiding ISO process to file issues will speed filing issues. Normal website to register & login will allow simple submission

DB will be on open-std? → no. it's hosted 'elsewhere' but is a portable git repo, will be mirrored on GitHub & other public visible places ↳ primary GitLab on GWDG

5.2 TS 18661 pts 4 & 5
part 4 (N3180)
augmented arithmetic moved to new header
type generics have simply been removed following guideline to not introduce identifiers; also unclear what some types end up being
software impl is available by Jason Reedy
CB: not convinced by library headers defining unaliased tag types as a design decision
PK: division funcs
RB: there is not currently a rule for this, good time
AB: unspecified types is convenient for impltors, exposes existing functionality
RB: in this case it does fit exact action → sure only generally
PK: div func doesn't forbid other members
Jens: true in general, may be padding etc
PK: could specify example if we want
7.1 would be the place to change
CB: _t types don't fit here; should be a typedef
RB: no personal pref. → don't want to set this prec
↳ do not want to flipflop though
Me: adding more names isn't a blocker
CB: should be minimal & not require tags
Jens: w/ auto we could avoid naming the return type at all
CB: please consider precedent seriously. don't like not having style guide for the stdlib
SHM: why did we shift to tags?
↳ we had 'a' struct; decided to name it
AB: _t is reserved for POSIX
↳ did consult w/ AG about it; no reply though

Decision publish part 4? 8 +4 / 3 / (Yes)
12 3 5

& part 5 - addressed sugg. chgs
same invention adding Constraints to library
Me: like precedent for

part 5 9+4 / 0 / 6 (Yes)
13

ACTION RCS to submit both drafts to Bill Ash / SC22

5.3 The Charter ...
nothing we say here will matter. no text to approve!
C2Y will be less than C29.

- relationship btw vendor & Std
- extended constit.
- nature of Prior Art
- not keen on Charter

Eskil our job is harmonization not design
'wiggling' required of course
change Charter or stick to it; meaningless s/n
WG14 as a way to force features through is
wrong ; limit to the argument
we have lost projects like Linux that ignore WG14 & order vendors around
need to focus on ~~a~~ improving corner cases rather than feature-add
C not chosen for safety + new features
feel that there is language drift to the point of a fork de-facto
& the old language still needs maintenance ; users don't feel
existing concerns are addressed by the WG
want a deprecation mechanism for the language; will resubmit
want to work on dependability & document the behaviours
supported IRL vs in paper ; would like a SG
RCS : JHM's paper on versioning library?
↳ that's a feature for later
Jens: don't agree w/ core assumption ; some things needed
adaptation but where things didn't exist they were not
adopted - new things do exist in field & are praxis

claim that community doesn't seem correct. Linux is following
but following slowly -
FW: support Gok, SG is good idea (no name yet)
on invention, way back: went w/ pthread b/c Threads didn't
work - why did we invent instead of picking one? or for
auto - choose things as-is
Me: 'epoch' maintenance SG alongside the deprecation group
to guarantee forward stability, dedicated fwd-compat
CB: dislike the chilling effect of a Charter that basically says 'go
away'; strongly dislike 'trust the programmer'; wants a
proposal for a rewrite. ↳ so misunderstood for too long
JHM: on existing practice; actively harmful, committee
fails to actually do this. some was cleared up in C23, but lots
not yet discussed; things like typeof taking 20y to make
progress. vendors did the experiments & failed to feed back
about exps etc in 2002; 'great idea!' but no progress.
impls don't have clarity on their feature maintenance. dampens
will to develop w/o progress. Charter is ineffective at
moving things forward. lots of practice atm to adopt, needed
for real code. large pool of features to pull from now
RB: agree by disagreeing. impls matter; uses of ___a-t was
so minimal, it seems inventive. wrt. picking a version - that
role is from ISO, to build consensus not an industry winner;
in practice 'no invention' has been largely ignored. outside the
WG nobody complains - uses like stability & restricted invention
do agree w/ choosing existing practice, needs someone to bring
something fwd. if uses & vendors have sth they like it should
be brought fwd.
JM: interop is not in the Charter & should be. need a space
for non-obs, for linking, linking to other languages, important as
a glue/comm language; this should be a core feature, inc.
ABI level; should not leave to impls., define up-front.
risk of new features is very contextual - building on well-und.
features is safe than building on new things, depends on compl

and interaction: how to get high quality specification? not enough people doing detailed tech review of other people's papers.
accurate integration - into the draft not always imported right, maybe need to provide diffs against the WD as part of proposal outdated changes etc. things we need - stdbit.h fits well, packed structures, portable versions, vectors, etc - things that suit C's style. safety & security needs work to make easier to do things right - managed strings etc. low risk of incl.
CS: agree about value being in interop & underlying; don't think fair to make vendors lead experiments, no longer true (need to maintain), users expect portability; less desire for vendor exts & need port. don't have a portable lang. w/ too much choice - Cmte must set an impl direction. becomes sth useful beyond small lib
AB: no two of us agree what 'invention' is anyway
 want inventing unifying solutions - exts have rough edges, we learn & correct the mistakes. do not want pevese incentives. vendors will benefit their schedule & not users if we unconditionally pick this
need room for corrections
RCS: we like C, C is not going away; never fallen below #2 TIOBE. Charter should be for WG, not for C; caretakers or stewards of the lang. do want to promote our work. champion.
- will never be t/m/t safe. not controversial. despite NSA etc attacking this record. there is money at risk. can improve safety & security without 'fixing' it
do we want C to evolve unguided or to have roadmaps? at the start of C11 there was a plan. didn't follow it. volunteer org. can only do limited things. what gets done is what volunteers want.
RB: strong support for JM's points. need review. put in things that are solid. no broken basis. vectors, simd, etc. are good but need volunteers. had a TS, died b/c lack of vol. (Intel killed it but nobody else picked it up)
beyond stability, interop - btw languages & btw C versions too. common link.
A.Bch: balance risk & oppo. seemingly easy things not easy.

(7) vectors!

took 22 years just to fix 'bool'. smaller, more papers should make review simpler. want a direction - C99 adds VLA, C11 pulls back; no good to reverse directions like this.
want clear distinct. b/w core & library - core has higher burden. would have liked BitInt to be non-core

Esk Linux finally approving fwd-decls - not much movement should consider having our own reference impl of the std lib we don't break compat. by changing code; prose is a long way around. agree that invention is needed to harmonize. skittishness around large features results in paring-down becoming useless ('constexpr')! should start SG for vectors etc. to force a schedule

MV agree w/ Joseph - interop in the Charter. new requirements do not allow to stand still. vendors should be involved & do the real invention. better process.

FW 'why C, why not Rust' - i like C, it is simple. adding to it makes it less attractive. features are offputting. prefer stability lack of typesafety is not true w/ better analysis

Ori don't stdize existing malpractice. don't know what all features or dependencies are - not obvious up-front. we need to invent but to know we caught everything in so doing. involve the vendors more actively getting directional exp. 'Austral' - understand small set, intuit a large - cannot intuit in C. want the guard rails but cannot get big lang without them b/c disastrous.

JHM not given that vendors will play along - GCC refused to comply. Clang need a proposal before admitting patches. no room to do stuff. vendors aren't actually interested in pre-Std stuff. when Intel lost interest in Cilk this killed it - contrast b/w vendors in willingness

CB agree, vendors do not have any of this at heart. prob. w/ whole model? chicken/egg situation! vendors defer to Std defer to impl

HK agree w/ safety & trust; work on public image. C++ did a lot of PR a while back. copy it. want SIMD
agree w/ Joseph on needing review.

AB wrt Clang - can guide about ext. process. both easy + hard
#embed has much user love - uses love same things.
nobody cares about C++ compat any more? not in our best interest, WG14 given all responsibility for this.

5.4 n2948 args outside main()
CS: getters. mostly for compat w/ other langs - Swift, C++ rely on libc to get at these but don't have access to main() necessarily. in Swift, CommandLine is globally available
having to parse /usr/self/cmdline is obtuse way around; on Windows need to re-parse whole thing; Rust injects calls around main
can't easily avoid teaching pointers to beginners
can't really provide well as a third party library solution
AB this is a very common user question & point of surprise esp. b/c lifetime
CS this is the exception among languages rather than the rule
const correctness involved b/c this is the safer addition, prg that does use mutable argv not broken nor encouraged.
something simple held to be best addition.
A.Bch cannot convert to nested const in pointer type (in C), should change this
PK argv is how C&C++ expose this to uses; this is 'good enough', init makes it clear that this is needed - convenience is not justified by feature addition. maybe if didn't exist
RB not strongly opposed but agree. for freest, does increase footprint
what about the impldef third sig for main()? seems ignored
↳ if your impl supports this, it can also pass that thing globally
do need to check & clarify this for next revision
RB what's the intent for other signatures? like one-arg?
↳ will not design on the fly, will review
Me this refers to things that _start demands to exist, so lower complexity than other new stuff
Jens what if user mutates argv?
↳ supposed to be very thin non-copying access - same identity & pointer; if they mutate it, it will be reflected here too

not considering modif. of argc - local, will not propagate, not ptr

Jens don't use the naming scheme.

↳ would be nice to be in stdlib, but will use w/e name told to.
don't care about spelling. thought this was allowed.

JM agree, need impldef cases handled; impldef better than UB
no mechanism for calls before main; agree on naming. only Ext

AB don't care about name. want in stdlib.h
motivation is mostly shared lib code - forced to have APIs they don't want to pass the data
returning a struct type would address the impldef sig issue by matching fields

Ori sympathetic to other lang ons of this. do they allow insertion?
taking params is good design to allow passing more flags to libs that weren't in argv. Some arg parsing libs do mutate argv, moving flags, decr argc, etc. then again some libs hack this in anyway.

Eskil does this work w/ multiproc DLLs on Windows? not clear it would work?

Opinion in C2Y, along the lines? $\frac{6+4}{11}$ / 9 / $\frac{2+1}{3}$ (Yes)

5.6 N3183 scanf undesirable UB
want to remove hard-to-handle UB not under user control
some edits suggested by JM
no prob. w/ chars, maybe w/ non 754 floats, usually normed
integers should be defined even if value is unspecified
trying behavior in musl & glibc use strtoul & clip value for shorter vals as-if by cast.
saturation originally proposed but burden may be too high after all

PK don't like extra wording for float - just use HUGE_VAL, does this like it otherwise. unspec is fine, not UB though. no need to heavily burden error case, just remove UB.

↳ possible FP reps w/o infinity → HUGE is for this purpose or NaN

CB this is a good idea & what we should do - want extended to atoi etc. most users never check Errno etc - value should allow users to handle do not think perf of error path matters

JM want HUGE, comp. specs that depend on rounding mode - normally used in default mode. should imitate strtod with adjusts for typ - define by ref to the funcs with precisely def. results

DB a lot relates to strto_ - well defined, can this have 'the same' beh. & errhandling? atoi etc should also align. not yet conforming for float there though
↳ want to add in second step, sim to strtol

JM if atoi was consis def, want a fn like strtol but w/ direct val not a cast b/c of dif range

DB guidelines dislike ato b/c less well def. & need to handle

RCS prefer indet. val to max/min, want to elim. trapreps so not used here either.

MU preferred design is not to touch value in error case - let dev fix it. tool can detect uninit reads, etc.

CB devs won't jump through hoops. unspec invalidates existing code which will check range already. care abt. regularity of library don't buy MU's arg - devs do not run in San, do not expect UB, should be correct w/o handling

DP error return in table isn't being checked anyway?
↳ diff. b/w Std & observed behavior.
CB: people won't check, not shouldn't → so it should work even if they don't

DP do a lot of code review; common beh. is not to check anything & not inclined to. so persuasive.

Opinion sth along these lines? $\underset{17}{14+3} / \overset{\frown}{3+3} / 0$ (Yes)

CB: we need code examples to make these decisions
JM: should be conv. other fns?

Opinion prefer std. based? / / → not polled (Abs)

DB cannot answer w/o looking up stuff
AB just want UB gone
RCS next vn. will need to consider these questions

Wednesday

5.6 N3089 _Optional
w/ thanks to AB for help

slides were for ARM intend pres. ○ what did colleagues think?
start w/ positivity; 'wears well' quote as aspiration
intended to be small & easy to learn ○ teachability
examples of doc. of boring nullability ○ opp. of checkable
time wasted on assertions, negative testing, etc ↳ 'kinda', negate
[static] not used in practice; fails expressiveness → ○ oversell?
doesn't currently work w/ void or non-params
attr-nonnull is even worse because of indexing etc; still not locals
↳ (totes fixable)
Clang has even more kws, qualify the ptr not the target
better than GCC at least ↳ ○ prefer to q. target ⓨ
'references' ... 'no'. (○ not actually enforced in C++)
↳ needed for operators per Bjarne
↳ greatly dislike the implicitness & syntax, support DRU syntax
Bjarne dislikes DRU; don't reconcile; not designed to rec.
thought experiment cot. references? not [static]. input from GvR (!)
type annotations are an interesting path - we said C could not be typesafe, but Python did it via annots. pulled in by 3.5
↳ in Python everything is already a ref → ○ not keen on name of non-monad
source of name _Optional ↳ ehh, Maybe
therefore: a qualifier, on the pointee.
works orthogonally w/ cvqs as-is
- rp to warn when used as-if non-null
- exception in & to drop qual ○ ok w/ this
demonstration of warn-ability
like Python, no new syntax for checking

good to think abt extending. <u>Does</u> change types in existing
code - all current fns. are typed as non-null implicitly
↳ this is the same change as adding const was
↳ that was <u>hard</u> to pass socially

Oi what about <u>voidstar</u>? special case for *& there?
↳ diff't rules about this in C, unsure...
would like it tho ↳ (yes)

Me (bulleted questions)

CB expect disagreement over this despite <u>having</u> impl. it
↳ another way different tools can give diff't warnings
maybe expectations will converge

FW 'i don't even use const' - changes semantics. but don't want warnings
↳ your code will carry on being correct ASTERISK w/o it
and you wouldn't get warnings
agree w/ Bjarne, don't want to force people to use new things

Oi ... as long as you don't use new fns. w/ this

JM echo - get N-doc for slides ; <u>memcpy</u> etc - should these admit nulls for zero? (special case) ; not clear when migrating what intent of old decls will be, not obv other cand tho

CB no strong feelings abt. other folks writing styles ; defensive is OK
probably favor relaxing it to reduce UB, but not to add _Opt generally
◯ that would be a dependent type
we already had the prob. w/ const ; can't always convey intent in C

MU like it. (Q lost)

CB C not only lang w/ lack of analysis ; should a non-null discard information? decided against, confuses the issue despite superficial simplicity

MU people hated const but the effect is the point & the pain shows it works

CB you can stop propagation w/ cast, always

RCS comment from Herb Sutter - perceived hostile to C++
(didn't show a C23 attr syntax example
↳ don't see mindset as reconcilable
not expecting attr syntax to be portable ; perceive as edge cases

○ pls submit slides?

in comparison to Clang: more orthogonal, more consistent w/ DRs
can be used consistently inside declarations if needed
argument for pointed-qual is that it propagates properly
  unlike _N which is not really a qual
  exists in signatures, obj-quals do not
Clang does check contradictions in _N but this isn't normative
(restrict is already broken)
'what a qual is' → about access, not value (tho. K&R call 'long' qs)
opposed to restricting the value range
  clear analogy to const which can be ignored * in a correct pg
works well w/ array syntax too compared to _N
func ptrs?
  ↳ can be typedef'd
    diverging permissiveness ; not a useful UB though
why not _Mandatory?
  ↳ better for interfaces usually expecting a valid object
↳ unusable crap

⊘ implicit conversion doesn't work
  ↳ this way around
  ↳ covered, irregularities seem
'an ugly programming language will work' → needs elegance
conversions discarding _O are an issue, don't want casts to ever
  be the solution ; *& building in the conv ○ safer than cast

not interested in #inc. solutions
tested by use at ARM ; checkable by SA (unlike casts)
changing the other derefs was much more complex
& is already 'special'
Migratable - an ignorable macro will work (asterisk)
  most functions can change in-place ; affects callbacks
○ clarity of stdlib !

people no longer in the room have
abandoned C

AB ~~on~~ Jens- like it! good idea, esp qualif.

mostly aesthetic but both reasons

SC do like idea but not that it doesn't indicate non-null
other langs do do it the other way, like restrict ; confusing?
cast &* - result is not null? most code cannot check this
right now (really?) used by offsetof/cont-of

~~AB~~ CB - std. these to not have UB

AB generally support std. something & unify praxis
brought to Clang & rejected ; did not like putting the Q on the obj, believe it to be the wrong level by multiple uses
the conversion arg. makes sense but there's a reason why val-conv does what it does
concerns about diagnostic properties apply to this too - to detect inside of if requires flow-ana
↳ not keen on shape

Opinion add sth along lines? $\frac{10+\overset{5}{\cancel{4}}}{15}$ / 0 / $\frac{5+1}{6}$ (Yes)

5.6 N3184 format spec for floats
normally adding scalars adds printing too - exception for bools
proposes adding for decimals & _F8C etc. also adds for scan.
problem of running out of characters in ASCII
taking precedent from int not possible in general b/c greater variance
but can fix to IEEE types ; not normative though
diffc to print correctly & fast : people fail, stdlib necessary
libs supporting 128-bit already have the ability so it should be named ; a single spec can only print 4095 chars
this is often not enough, even for %s
the limitation is in snprintf & shared across all funcs by it
should lift the limitation
must avoid temporary storage but impl can stage the output

Jens tried to add ints in musl ; difficult b/c don't distinguish lang & lib support of types, macros not fine enough - more macros to feature test pls
don't like overparsing printf

Bch: print is sc of ers but still simplest way
agree abt non distinct btw lang/lib impl - wider topic tho

Jens could still have stdio feature macros

Bch we do have tests for the types
↳ other way around tho - type w/o lib support
user has to mimic by hand

AB not strong opinion but conc. by choice of spec - not same btw pr & sc - wxn, wnx - why diff?
bitwidth being lost is preferable for consistency

JM would have to be wxn - wnx conflicts with int case
on lib/lang support, we have STDC_VER, very limited
does it update per-header? not complete answer though
diverg. on C23 already, e.g int128_t w/o intmax_t
not necc. communicated & implies lib support
should not block later adoption but need

Me toss out format strings entirely

JM break support for intl. by doing that - lookup of entire reordered string would need to be preserved

ABch defence in paper

CB how did this arise & why was it OK?
↳ possible to use sth. nonportable like Q

RB we didn't add b/c to_string exists instead (char-limited)

RCS cannot do Decisions w/o working draft

Opinion sth along these lines? (all) $\overset{8+4}{12}$ / 0 / $\overset{8+1}{9}$ (Abs)

Opinion raise 4095 char limit? $\overset{7+2}{9}$ / $\overset{2}{2}$ / DON'T KNOW $\overset{7+3}{10}$ (Abs)

PK this could req impls to support huge objects
don't want this on hw that has less memory! can't create the buffer

Oz need a buf for bigint → not true

CB agree w/ Philipp, either both t +s or neither

MU do we know how anyone else solves init'l? 'badly'

5

N318~~4~~ Simple TU Init

dynamic init of static stuff - doable, always nuisance
dep. b/w TUs (SIOF in C++)
GCC allows ctor identification & ordering, with explicit nums
like line nos in Basic
already have call_once & on-exits ; not that great for dependency
b/w TUs because of the once_flag
proposed per library evolution but need compiler magic for ctor-ord
DEFINE/DEPEND transitive property across TUs ○ could compose nicely w/ if-decls
symmetric exit handler without callback
if funcs are not entered, nothing gets registered - good for optional lib components
also uncond. variants when wanted ; weaker expressible
?1/0 ref. impl! nice & short

MU mentioned GCC - do we need? → do need for STRONG
works for WEAK, prob. better tho
Me ONCE_DEPEND wakes on explicit call post-main? → Yes
opt-able
DB GCC thing is for libs, which are out of scope?
↳ diff't trans units inc each other, both have state, want to ensure order
initialization before main no? → dep. on each other
can't you wire it in from main? → not from other TUs
so. arb. complex init for libs, not called from main? → Yes
Jens call_once already does this but w/ user object
DB rel. to libraries? prop. is devoid of lib concept, equiv to GNU?
↳ yes, specifying what & ordering of what should do
AB wrt libraries - across static lib boundaries, is there magic?
↳ no, no magic. impl has chosen some mangling of init fn
extend symbol w/ normal linking

Opinion sth along... 7+1 / 3+2 / 5+2 (Yes)

no consensus - proceed as a CRFI instead

* June Meeting will be held

5.9 N3189 literal constants
need to separate terms better: lit for gram, const for sema
mostly simple, some ambig.
RCS (is constant & constant value the same?
↳ sth prepro this is weird ; numbers aren't really numbers, but used
not optional inside #if
characters make it even more confusing!
text will be reworked
RCS great.
FW literal into glossary? → don't think so yet but consistent,
(cl 3) we have the distinction already, unify
DB slipping into multibyte - what is that? an array? code point or unit?
↳ don't see it implies that
↳ not part of this suggested change, think u8 implies ASCII
Jens not suggesting to create a normative change
a multibyte character is a number not a string
DB 'how do I put it in a char'
RCS std has 5 conflicting defns of 'character' - tried to alter it but is a whole-doc change
invite a proposal like this for character
JM 'literal' is in 2382 - lexical token by itself
PK UTF8 chr shall not contain wide values, only u8pfx
FW 'narrow' is to mean 'single'? → Yes
Jens confusion w/ 'integer character literals', but that's all of them
RCS see also p2749r0 (CJ)
JM POSIX defns say 'byte' when they mean that
DB POSIX also requires 8-bits
CJ all that work was done for C++23 as well; consistent lexing

characte still not meaningful in a technical ctx. No need to be inventive
as the lexing is the same, suggest 'ordinary' used in C++
also 'UTF8' & 'narrow'. All content is fine, just prefe consistency
all this is unrelated to 'characte literal' anyway. C++ did allow
multi-char w/o implying multibyte - dt. concern

Opinion changes along lines? 24 / 0 / 0 (Yes)

N3190 prepro enhancements
want direction. goal is language-indep. pp., lack of sync btw
C & C++ ; would like to split off own specification for IS to cite
start by collecting extensions
drawback of COUNTER is eval order becomes observable
MSVC does weird stuff here ↳ still useful for local uniqueness
functionlike macros are just better as builtins
expand_dec etc is more fundamental; again, prepro knows
how to do this, trivial to export
many possible applications for new literals
O implementability char-ifs
some divergence btw. langs & vns with
spacing & suffixes, esp C++ suffixes ; space after prefix causes the
same issues on the left ; prefe this to be a diagnosable rule via
composability w/ embed
params for #include ♡ ☺ loweffort once #embed is supported
#bind is prepro-scoped
use cases for expanded include? → [nodding b/c do]
↳ maybe regular uses less so
#expand fixes the way #include is broken
↳ this would make #define stateful O

FW exact names? → exact. What do compiles do?
CB great to unify praxis, but; Bjarne & C++ don't like macros
does C++ even care? → well they're still extending the prepro
& ideas from there do come to C, so should sync better

○ COUNTER SQ?

pragma once? → voted down

OB love most of this ; prepro description is getting big now

Jens want a feeling for how far to go ; have started the separate doc

JM L1-PP seems reasonable, consist impt. uses outside? not a good idea, closely tied to these languages & lexes, space rules etc.

↳ well it is → well it's a bad idea

↳ want to be conservative in what we add, loops etc. maybe not

↳ all implable w/ OPT

↳ explicit looping etc - more skeptical - in general does not propose polls ; should not ask all in one go

on #p ONCE - pain to define, what is 'the same' etc.

we have a praxis solution

Ori C++ trying to remove macros & didn't want in modules, though supported atm ; prepro 'cleverness' is a source of pain

hard to debug ○ reduces this!

prefer explicit codegen

Jens some of this reduces 'cleverness' - no more NARGS handling

Boost etc can be simplified, less dep. on diverging

Ori would #embed be broken by B?

↳ you wouldn't use if-empty w/ that

would like zeros to go to any object type

PK burden on users & implementors - must weigh cost

don't see values in EMPTY - expansion in #inc is also problem

Me [bulleted comments]

DB don't want it to be a prog. lang or become one

want a generic prog review → not plausible

JHM many users wanted offset() — after it was added

Clang has the patch, want to est praxis

users want _Pragma embed - after it was in

favor predefs & literals strongly ; impl has the info for this

CS good ideas but scary to touch #include

desire among Clang users for this sort of thing - esp loops/rec

do need feature-focus

NO POLL

N3197 chartype arrays
first for c2x, N2014 by Kayvan Memarian
reviews users, vendors, etc for their understanding of Std & behaviour
'can I put an object in a char buffer' — most users think 'yes'
and most of the rest 'probably'
App devs say <u>YES</u> & rely on it
vendors/Std say <u>NO</u> — clearly total divergence of opinion
people do this so the Std can define it as valid
N3197 therefore seeks to allow a (non atomic) char array to act as
a byte area ; adds byte concept to glossary
amends the effective type rules
allow for atomic_init ; change alloc fns to define in this
MU compiler writers don't actually exploit declared storage - believe
this is already compliant w/ all praxis
◎ string vs byte typing

FW strongly favour this ; would make it possible to implement malloc
char or uchar?
↳ other direction currently allows all three but can review this
↳ byte type with special status?
C++ has that but it has special operators ; type punning?
same rule as before

Bch <u>huge</u> step in the right direction ; overwrite?
↳ Yes, changes the ET

MU though not always correctly impl
JM 'unqualified' byte type? wordings contradict e/o
could do w/ discussing C++ semantics
RCS we looked for wording but couldn't find it
Jens C++ is much more conservative, 'new' etc
Ori improvement, code assumes this is valid ; uint8_t? do we want?
hard to opt?
↳ aliasing, ext. int
MU we do not know of that → could work anyway
PK would allow any char array to be used as any type
archs w/ memory addr spaces in EC-TR - 8051 doesn't

allow atomics in all nested addr spaces, only 256b

Jens would it help to distinguish non-strings? agree, atomics are hard

Ori what happens w/ malloc? → not used... user specifies addr space

CB not sure what point of calloc, malloc changes is? use changes?
doesn't seem to do anything. worry b/c GPU addrspace can be
oversize, need non-ptr types ← may damage such practice
↳ seems out of scope
addrspaces are not well covered
part of GPU space is 1:1 mapped

Jens we aren't changing the ptr or anything like that
↳ prefer malloc to stay as-is

RCS intent is not to change behaviors, only to unify terms; want the
same effective type rule for all types of memory, simplify

Rch Std only talks about the generic addr space, others are TR
should not ignore part of Community but also not allow to block it
huge problems from allowing embedded locks already

CB to be clear this makes my company happy

AB casting is UB in C++ - placement new used for this instead
explicit lifetime control.
100% in support though. widely used in praxis.

Jens placement new anywhere? → yes, char & schar too

JHM echoing AB - no need for ET rules in C++, lifetime is explicit
ET is C's equivalent to that w/o adding an API users won't use
is 'conv to voidstar' CB's concern?
↳ appears to bless allocation from an array, not mmap etc

JL not to bind bytes to 8-bit - 9 + 16 are in use
this has a definition that is fine, addr not octet
prefer to unbind bytes & chars? ○ ET etc

Jens this binds in both directions

JL could we add a new fundamental byte type?
↳ C++ has one and it would be different

JM multiple addrspace issues: not in scope, we could integrate tho
(○ agree w/ this); decoupling bytes? already overloaded
term

AK only real prob is lockfree atomics ; could an exception be carved out
Jens or feature test? ↳ or for hosted only, use cases are there
Ori this is like malloc, that has it or doesn't, same
RB Martin saw bugs? → NO   Me (ETs)

Opinion   sth like this? $\overset{13+5}{18}$ / 2 / 1+1 (Yes)

N3186 Init & Eff Type
aim to make ETs easier for users to control
communicates new obj. identity to the compiler
includes FAM structs
much like the other paper only allows ET where already allowed,
RP to erase existing data
alloc fn. is a bit like 'new'. the compiler should treat such
calls as a new object every time
'flex' is own API (lot of params...)
the length member may need to be mutable for most flex structs; not so
with this, known size
Bch why only ints? → first arg is length, only that one
JM disc. on Reflector - needs UX with the feature to judge friendliness
what mistakes do people actually make? not experts. should fit more
as a CRFI ; refs to memset_exp are odd, limited niche cases
not related to this
Jens wanted to force calls not to be allowed to opt-away
memset can be, which was a mistake ; want old data to be lost
GCC can allow this, or just force noinline ; intended to guarantee
state, not to target efficiency
JM want opt-away to be the common case, not forbidden
if you're not overriding m_e is rarely relevant - hardening, not
semantic ; user should call it directly
Jens want to avoid surprises later ; users should treat info as lost
JM not the norm though ; like = expect to be fast, non-ortho
TJ realloc use case? lose the original ptr, can leak on fail
↳ no realloc interface except flex

AB so how to model that dependency?
↳ by name only, through ONCE_DEPEND
Ori set of macros creating calls, explicitly in main? → Yes
AB static lib; have O_D, do not call any APIs — will it
be skipped?
↳ will not happen unless DEP is hit at runtime
how do i know local names? → matter of API documentation
this isn't a header feature? → it's not an internal detail
not replacing gtk_init etc?
AB may have misunderstood
CB 'ew macros' ; concerned by naming & namespacing
strange ONCE_EXIT naming
could be split, load/unload & usability improvement
Jens naming can change. we already have once_flag
'macro is my thing' ♡
could be integrated as funcs/builtins, want to be sim. to at_*
leave lots of room for exts, like dumping dependency graph
CB pref would be to split, value in the second part
TJ how does this work w/ OpenMP etc. with diff kinds of
boundary?
Jens this makes a fun call into a library with the content of the block
this is the normal model
TJ in the logger example this touches outside state?
↳ not intended to be a global lock, it is a dep. ann
JM optional support for pre-main needs 5.1.2 changes
↳ yes, if we follow up. direction for now.
can we call exit() before main
JL nice to have in-lang namespaces...
you impl this? can we see? → using the Mod C extension
will share example projects but not all portable
RB love it & that it works today
looking at STRONG, not clear that it can; external symbols may
collide → this is part of the ABI, could use reserved
must be consist. across TUs

similar effect to a realloc call, nothing new or fancy; change only the length mem of FAM

CB don't see value esp. vs complexity - everyone else writes iteratively
purpose? pool mem?

Jens give a simple way to change the ET of sth. so an array ptr becomes guaranteed

CB so is this like wrapping malloc? who uses this? don't usually init like this, use loops

Jens calloc is not enough, only zeroes & no named type; better way of doing

Ori nontrivial example of usage? → no... split from other papers, like ident for FAMs. difficult for user code to do & stay safe

AB share JM's concerns about mx - cannot opt at all, inc reader/coder
users will avoid these like that, even if there's value in guarantee

Jens not convinced init is the bottleneck

Opinion sth like this? 1 / 6+1 / 9+5 (Abs)

N3187 clarify array length exp/spec
problem w/ understanding when effects occur - aim to simplify
eval of ++p depends on eval of [n] - unrelated seemingly
very hard to get right
C23 allowing extended constants makes it even less clear & no way to know what happens ; try disallowing side effects
● what about the composite type case?
can also constrain fn. calls w/ unseq.

PK consistency - forbid side effects in alignof too?
↳ never evaluates so prob doesn't happen O symm

JM issues on Reflector; side effects are generally confusing, except when wanted; mutability all has this problem, not lim. to VLAs
def. not unseq in example: a VLA ties a size to an array -
* doing so directly in defn reduces chance to lose the binding
nested complicated features can always be confusing regardless
realm of guidelines

Jens eval needs to take place for lval conv; not good when there's an effect

don't agree w/ PoV, people don't expect this to have problems
good idea to warn users
↳ CVio is just making the warning mandatory

RCS don't try to change ideas in the room

AB hate the feature, surprises users a lot; impl test cases mostly through whack-a-mole work. allowed exprs - RP to diagnose calls? back to being surprising. this will invalidate a lot of docs w/o fixing the core problem.
↳ so vars only?
Yes. mutating ops are the problem. → prefer 3rd vn? No calls
we could allow constexpr fns ○ [unseq]
usually don't want to break

Me MISRA; like opt to remove UB; → whole UB bullet

CB banning sth doesn't stop folks doing it

RB don't like dependence on an attr in v2

Opinion sth along any lines? $\overset{11+2}{13}$ / 2 / $\overset{3+3}{6}$ (Yes)

directed consensus.

N3188 Array length state
adding sugar to query the prev-established state. many usages
identifying that VM types are actually the same
allows user access to a state that is already there (in typesys or runtime)
for VLA the AM-state is reified
size of an incomplete array is constant but not linked semantically to the visible name; in defining TU there is maintenance effort to keep these externs in sync
○ use of desig-like syntax?? prefer a diff't formulation [extern SJ]
enables analysis of link & size info
not however intended to introduce new ABI - this is two symbols as it was before; purely additional sema info
also usable for follow-size params

PK do we need the dot; size already in scope? not normal

VLA syntax? size in scope etc.? require same decl
Jens doesn't work - w/o giving length the TU seeing decl-only would
not know dot size, must be a linkable identifier
must indicate it is not an expr to eval, |
Me propose 'extern' kw inst of dot
MU also don't like the dot, should introspect
agree w/ PK the external use case is most useful & may not need dot
↳ this would be too ambiguous, is it defining SF or what
this changes the meaning dpt on presence of other decls - diff't concept
SL love it ♡ also want dot though - consistent w/ Linux, precedent
JM cmts on Reflector; dot syntactic complexity wrt. namespacing
much simpler w/o; SP for this by feature w/o syntax
structure case impt; parameter case diff't important; externs seem
less relevant
CB like it, not sold on syntax; dfo keyword though, inexpressive
maybe should vary w/ context - sensible in struct, not so outside
dislike implicit declaration by naming things; meaning should not
change w/ context
↳ convenience mostly; not needed except for param case

Opinion sth along these lines? ${}^{9+1}_{10}$ / $2_{3}+1$ / 2^{+3}_{5} (Yes)

Jens any objs to specific features?
Bch don't use the colon → ignore syntax for moment
JM don't want the return types part, underdeveloped
Jens idea is to have same contract as params - must be identified in params
inverse of [static]
↳ comment 16 ; how does caller know sizes? rules for [] → * apply
MU like return types, needed for
Me/RCS - what's the poll here?
CB so type is based on sth subsequent, not yet in scope
if not a prob why other paper?
↳ move to that one

N3207 param fwd decls
[discussed thoroughly before] fwd decls are less convenient but
has most consistent & clear semantics
arbitrary decls here mean additional use cases exist
fixes to technical issues ; C++ compat doesn't matter, this is a
non C++ feature anyway

RCS if the decl is not used, is that an error?
↳ we did allow that & don't any more ; example given of only
declaring a type remains allowed, could prohibit
↳ want to differentiate errors by accident from unused feature

Jens at one time only wanted to allow integers here
↳ that was for rapidity ; usage is more broad in, attrs & sizeof
no reason to restrict it even if not useful obviously
fwd decl a struct allows sizeof(member)

AB Clang has been investigating for attrs ; tried novel design w/ just
using name late ; delayed parsing, low burden
this doesn't generalise to decl attrs, usability difficulty

CB want to support this but misgivings abt allowing 'unrelated' decls
here. what is a param list - are these decls, or just 'used later'
this looks like a semi-sep list, w/o prec. resembles 'for' but
w/o structural rigidity ; one semi or none? emph. not a
list of equally treated things

MU do not know why it looks like that - existing practice ; GCC
does allow commas too, even less clear
↳ commas indicate ordering, these aren't ordered

Me we rejected Clang idea for C23?

JM decl before use is a general C principle ; once you see an id it
enters scope, no lookahead ; fits w/ decls b/c not used there
referring fwd. is a C++-ism ; inventive syntax is possible too
so long as decl is actually done

RB solution to a problem we created & asked for ; this solves it

CB not allowing this is a valid design choice

Bch do not want the size_t restriction - rules should serve a purpose

(already polled many times)

Opinion want it? 9+1 / 1+2 / 4+2 (Yes)

poll on allowing non-params

Opinion non-params 7 / 4+2 / 3+2 (Yes)
↳ guidance

CB choices seem to come out of thin air...
RCS develop more between the meetings
MU want any solution & don't care what

```
void f (_Param a, b, c ;
        _Param d, e, f ;
```

N3192 Seq. digits

JL why not fix charset? AB - SDS prev.

Opinion add to an SD for next meeting? 15+4 / 19 / 0 / 1 (Yes)

N3193 octal
CB don't like magic numbers but this picks a fight w/ Linux
Jens prepro & printf unconditionally interp as decimal ; consistent
JM also escape sequences - delim. only in C++, consider this instead
RCS generalized 'base' prefix? for arb. number?
JL like that thought, but 0o is worthwhile to do in one char
obsolete doesn't mean removed - can stay forever
RB strong agree ♡☺ errors in user code & common pitfall
PW change existing code? → not realistic to do
DB can diag. now but no alternative syntax to use
Oi def. need alternative, much use ; careful abt no. warnings
SHM think obs. is good - want warnings earlier
signif. of leading zero is a consistent pitfall

Opinion general? $\overset{15+3}{18}$ / 0 / 1 (Yes)

Opinion imm. obs for O? $\overset{8^{+3}}{11}$ / $\overset{3^{+1}}{4}$ / $\overset{3^{+1}}{4}$ (Yes)

ACTION SEACORD to add n3192 (hex) to SD

N3195 named loops

JL we have goto → two sides to the coin

'goto outer' → current feature

MW like principle but visually confusion → convention established

Ori Go has both features - users do find pos. confusing but not change

CB don't like goto & therefore like this ; K&R consider this one of the only legit usages already

JC favor but want arbitrary breaks from any block, sim. early return

AB favor ; edge case of inte?

consider allowing null? → already allowed

Rch C traditions - Thompson said goto is best way to jump

what about multiple labels? believe this is counter to Charter

PK expands std w/o benefit ; goto works already for this

Ted should be valid as a goto target? → it is one

CB Charter arg is not compelling

AB multiple labels? → want to allow this

JL don't disallow goto-restart

MU diff't syntax? → divergence

JHM like, but don't like placement of label - not broken but confusing

avoid ambiguity btw intent, prefer) label: {

common use case

FW could just use defines for synonyms for goto that are checkable

Opinion add sth like? $\overset{8+5}{12}$ / $\overset{5+1}{6}$ / $\overset{2+1}{3}$ (Yes)

N3194

FT use of characters here

CB downstream impact on other langs

RCB what's wrong w/ :: ? → mess

Jens just do as GCC does

AB support & also following ; cost :: , namelookup in constexpr is ambig
closes door

RCS math notation? not practice, not balanced

DB not convinced by right-open ; discarding enum?
↳ potential change to enum semantics

JL .. instead of ::

Opinion in general? 14^{+5} / 0+1 / 2 (Yes)

Opinion right-open op? $\underset{6}{5+1}$ / $\underset{8}{5+3}$ / $\underset{8}{5+3}$ (Yes)

N3196 if decls

AB ♡ this, so much cleaner, really simplifies code

RB does this exist in C? → nobody to use it r/n

MU easy to add in GCC

PMc does it apply to else? → it does because C++ does that

RB stacked decls in elseif? → yup

CB favor but confused?

JM secondary block would mean it does apply w/ current rules to else
confusing semicolon usage in syntax

DB assign-in-if? → not the same

CB very common usage IRL ; rules are noisy

JM issue is when it's accidental for == - not confusion

JHM can we have many decls? → not in C++ version
not impossible

CB multi-decl? → works

Opinion sth? 9+6 / / 6+1 (Yes)

N3203 strict order

FW Ocaml doesn't specify

RCS for loops? → full seq expr

Jens side effects on the same obj? → defined

RB no portable code breaks ; good thing

BCh dislike it ; UB r/n means it is easier to reject, well-defined means
secondary rules are needed to reject it, UB useful here
↳ 'defined erroneous'?
▷ cannot generally be a constraint ; compiler can't see into calls
or lift into an order ; cannot ignore stmt exprs

MU like the direction overall ; tricky potential fixes to compilers
new code might break w/ old compilers
addition of ptr to integer may eval a VLA

SC generally favour dropping UBs - is this valuable for compilers?
diff't order of diff't opts?

AB performance changes as a result of the C++17 change? ○

JM more details for what did & didn't go into C++; what about only
going for the same subset ; semantic ordering better than L→R
in all cases ; rules here turn inits into assignments may be hard to
optimize (& this is not a C++ concern) ; order is documentary

Ori want more benchmarks ; order of post ++

CB already dist. btw 'valid' & 'good' code
don't think unpredictable behaviour is a good thing

JHM < NDC rant 😮 > C++ defined options for fncall ordering
b/c plenary objection to original L2R
based on a maybe
overloads for streaming & printing were originally unsequenced
support, would have brought it ; exception only existed for old tools
determinism is good for users ; C++ obj was bad decision
also stmt exprs even worse w/ variadics. avoid mistake

Opinion to pursue? 11^{+4} / 3^{+1} / 1+1 (Yes)

N3201 _Operator (Marcos)
↓
N3204 sema

RB don't want overloads but if we have it, this is better than C++

JM not sure div falls here ; practicality of pulling in all types, mixed types ; - everything w/ your new type - huge numbers of overload decls? feel like banded is going to be verbose ; either is a mess
↳ feel like 'not worse'

Osi some experience implementing this ; impl. convs make reasoning hard
conv can be defined to be in the operator or outside
plays well w/ generics, should be codesigned
need less but do still need some kind of best-match in practice
found overlaps

JN automatic type conversions in LCC ; can define with widest types & allow conv to that → same as C++?

AB cf [[overloadable]] - 30k uses of it, strong

N3205 - discussed w/o votes

N3199 defer

JM we do have vote from before to put into TS

SHM this is more concrete than previous versions ; captures, lambdas etc no longer problems to worry abt
cleanup is picked up by Linux kernel ; lot of praxis even calls itself def
↳ spelling too
there is no runtime char associated w/ it ; scoping is totally static reg. no allocation, contra Go ; no unwinding presented
↳ terrible design mistake requiring use side fix
and implies arbitrary allocation
requiring a secondary block leaves room for MISRA to demand braces
val-capture leads to double-free errors ; no object, scope works usually
no praxis for unwinding ; no storage ; GCC does require exception awareness for thrd_exit ; assuming this is for C++ compat

pure compile time features imply zero cost
even found and fixed leak errors in the provided example code!
freeing a dynamic list of resources is hard w/o this!
_attr_cleanup is also hard to use, easy to use wrong, e.g. free is rarely correct
wrt. human error - usually only some paths; custom alloc & free make resources harder to manage for SA
cannot change the value of return (unlike Java)

RCS easy to diagnose?
↳ Me: yes, it's a dead store

JL x is ptr? → can still write to ext state

JHM this exactly matches C++ dtor semn
jumpout and jumpin are not allowed.
Ctors & dtors imply heavier feature sets, make mangling

○ same as finally, removes defn of cleanup from create action
Sommerlad couldn't get defer into C++ b/c noexcept rules
dtors have a flow; reusable, but require an object w/o context
↳ filebuf just eats exceptions
object doesn't have visibility of the scope around use - must capture
personally want both much of the time - but want sth that works for C
'defer' is roughly implementable in C++ anyway
try/finally doesn't allow common scope - nothing in try is visible! get lost before cleanup

Qi like this more than before; enough uses already have established praxis, 'C as she is spoke'; unreachable defer?
↳ not run ○ scope rules

* blocker for TCE (thanks Qi)
in Go sometimes you don't want it on the exit path
→ D has a scoped error/success concept, C lacks concept
panic & recovery would be weird w/ it inside defer
these belonged to the TS

CB love this ♡ misgivings. ARM want defer. concern over usage
'one person's idiom is another's antipattern'

main use case seems not to fit, w/ 'make' - on anti-panic function?
concerned abt verbosity & UX
↳ something to not run on succ. pass ↳ most needed in opinion
not really elegant; no clear concept of this; we could add a kw?
CB generally do 'work' in a sep fn so early return is easy
RB removed 90% of objections; concern abt performance of ret-before needing a tmp space; considered other way?
↳ compat w/ praxis as driving reason ○ Sao does that
looks good, want experience in C so want a TS, hope we are
DB one doubt: defer attaching to block seems less flexible ○ cost though
↳ can be written as sentinels etc, or nulls
will potentially need 'if's - want to allow that kinda-thing
current sys allows 'take ownership' idiom better
very simple & predictable
'defer p && free(p);'
Me [bullet points]
Ori Go actually allows change of return value - don't feel we really need this though
RCS during C23 we decided to have a SG write a TS - didn't do so
do we want the TS route or to use the full dev cycle
MU need impl exp but TS was for sth far more complicated
Me will you do it tho?
↳ AB: as Defe, maybe; very on-the-line
RB: much more chance of a TS getting in
a TS can be lightweight & be ready by E-O-Y - positive
almost as strong as an IS but can be fast

Opinion develop as a TS? 11+3 / 2 / 2+2 (Yes)
14 45

Opinion for C2Y? 9+2 / 3+2 / 3+2 (Yes)

DB TS vs PAS? → _shrugs_

US, UK, Can, NL, PL, FR support

<u>ACTION</u> SEACORD to submit NWIP for defer

N3198 cond. unwinding
if we have defer, users want it to unwind
option to unwind, as C++ & less praxis for longjmp etc
propose to declare the amount of unwinding that occurs
would be compat w/ C++ & others
new small section added for unwinding w/ spec to existing lib
predicated on having defer

Jens for signals there are many different scenarios - sync & async, segv/error vs call to raise
↳ functionality considers equal but want wording for state prg. is in

CB not clear what users do w/ this? pressure on impls to provide, may make defer less attractive, not just code movement

RCS issue is diverging praxis - not a request to add anything, only to communicate it → exactly, expose practice
↳ response to user concerns, not necc. part of feature
↳ fear what users will do w/ this

JM flawed approach - property of other TUs, not the current one - not clear about kind of signal or masking, etc. Not useful approach

Me : TS toplevel chapter lets vendors ignore it

RB no provision for conditional? impls can sometimes recover, in some cases no 'partial' answer
↳ might need specific variants for each condition ; maybe out of depth
↳ we allow registering a kind of handler ; runtime check, or 2, useful

RCS elders regretted even adding signal handling

JC purpose is to indicate to users - impl by compiler or by runtime?
will compiler make it work, & will lib let it work ; who defines it?
intended to be lib runtime values, not compiler constants
akin to glibc & -fexc

Ori signals allow very little useful ; only extreme caution avoids broken code
really need to know how, esp. SJ/LJ
prefer to specify as non-unwinding

JHM there is a little wording for LJ already - CB maybe capturing practice is impossible

Oa default should be not to unwind

MU agree should specify; impt. use case is interop & don't feel this fits
may be disruptive

CB do consider what happens on segfault - defer can segf again, don't even trust the concept
accept defer if intermixable w/ other styles; unwinding makes them 'special' which is disturbing

Sens signals don't make sense; should have a policy for exit handling strong vs quick, etc; C++ sandwiching C code needs to passthrough in its own way
↳ no way a C compiler can help w/ this; need to provide C ifc.
have to let impls figure this out

Opinion put this in defer TS? 2 / 12 $^{+1}$ / 1+4 (No)

N3200 transparent aliases
'typedef but for functions'
purpose is aliasing libfns w/o ABI changes - indirect to diff't impls
this leaves old names exported by linker w/ new name visible to new code
existing practice of asm('label') feature used for binary-backcompat
allows upgrades w/o ABI breaks
implemented in Clang & tested: confirmed no ABI break

MU don't understand how this solves the problem
a macro can do this if it's a header decl? macro rule exist to allow decl w/o including the header; direct access is not fixed by this
dlopen doesn't work w/ this

SM suggest 'auto constexpr' for this purpose should be allowed, very similar effect w/ fewer consequences
↳ does get most of the way there, except for & operator; depends on priorities → or give it func's type, not ptr

CB cannot remove the suppression rule w/o breaking much code
don't know why this mattered

Ori dlopen ; ABI breaks is really about types, <u>not</u> constrained to one lib, but passthrough btw libs - cannot be fixed in just one place
handled out of language, by .so loading

<u>Opinion</u> sth along lines? 4+3 / 4+1 / 7+2 (Yes)

N3206 - Imaginary
Jim ~~CFP~~ have given valuable context
few but no feature outside AxG
Not aware of any impl that does this (GCC lies abt conf.)
Jim refered to HP impl, not maintained
imag. operations change domain on mul ; nobody supports it
macro 'I' is always complex, de-facto - now we have type inspec
seems like a dead end
prefe to remove ; could also remove from S6 & split AxG
for literals, the 'i' suffix is ambiguous in intent, prefe to be complex
very preferable to I macro
someone couldn't add imaginary today
corner cases in arithmetic
Me like suffix ; Imag is unused
RB probs w/ suffix, loses IEEE compat (for I * inf) → (Jim)
RB in Chapel, imag being added w/ full sems
RB (self) no customer eve asked for this or for suffix
Jens: import to get i b/c allows to drop the header entirely
we can separate more easily in C23 if more quotes
AB zero customer requests ; one user for Clang, 14 y/o - no rationale
customers <u>do</u> complain about the feature macro, requires collusion
which is hard to provide
DB find this very odd as a feature - what is imag type for? muls are useful
but can be a funcall ; polar complex seems missing ; awkward type
that doesn't fit
TS in Fortran, widely used & fully supported ; poor interop
Fortran allows mixed-precision complex [64, 32]

Opinion → remove imag? 6^{+4} / 1^{+1} / 8^{+2} (Yes)
consensus! to remove

N3051 LCC-style operators

believe the best option in-lang is to give users tools to do what they want; not to limit. New numbers, etc
use cases include fat ptrs, counted strs, s/o arrays - all v. useful
against only importing numerics - [] is equally important as others
against things like adding strings, w/ not commutative; prohib. in prop

Bch going further does depend on the mechanism - like hidden allocs
↳ not hidden. function can do whatever it wants, but not intrinsic

Ori this is a feature that pulls in others - indexes need to be assignable
↳ (assignable) → w = []
↳ more complex; classic is big-int, where you want allocs; which needs temporary-handling. much to consider abt. what is pulled in

JN several ways to avoid, or use a GC; not justif. for avoiding, this is solved by many other langs

JL not a fan; there are use cases but it is not obvious or searchable
already have macro magic; don't want more magic

CB have uses for this but we don't need to add this to the language
we can define more operations if we want
this proposal demands too much; 'must' overload call op? not necessary to pursue; == &!= is also what don't want to allow

HK echo CB's point that C++ is crazy; complexity is explosive
do not believe value is added anyway; how do we resolve the call? what is the complexity cost

JHM do want overloads, but not like this. emph that when we add sth from math/float, taxing impltrs to get semp right - this is why we removed imag; do not want piles of Annexes in a misguided attempt to keep C 'small': fixpoint, vectors, etc
let vendors do work once & let lib authors take over
also not interested in limitations of impls - _BitInt limits make it almost worthless
not a scalable approach; must defer to users to work

limits were needed to get it in at all, but fundamentally damage & users disappointed by ensuing importability

JC prob. with this specifically is the focus on free functions - doesn't work well; better to define a type relation to existing ops rather than eg. definable == ; diff. w/ ptrs
implicit convs are confusing at times; best set sems are confusing
requires a lot of design for C & haven't seen that yet

JN specific prop is designed to be easy to understand; C++ is overcomplicated & try to avoid by defining simpler rules; but the lang is becoming complex & we cannot do all these for all uses
impossible to design one spec for all use cases; this is more minimal
no problem w/ redef of == b/c only in new code. No interference w/ existing code

JL think abt how we can have libs do this instead, ~~of~~ fns; think this is higher level

Opinion in any form? 4+1 / 8+~~4~~3 / 3+~~3~~4 (Yes)

N3201 Operators (Marcus)
more flexible

RCS we can have both again in June → no vote is binding

CB someone can sell the idea better later

Decisions rev.
Actions reviewed

RB moves to Adj., FW seconds

Adjourned!

ATTENDANCE

pass me clockwise!

WG14 2024-01-(22-26)

NAME	COMPANY	COUNTRY
Alex Celeste	Perforce	United States
Dave Banham	BlackBerry.	GB.
Rajan Bhakta	IBM	USA; Canada
Philipp Krause		DE
Aaron Peter BACHMANN	Efkon GmbH	AUSTRIA
Eskil Steenberg Hald	Quel Solaar	SWEDEN
Jens GUSTEDT	INRIA	France
ROBERT SEACORD	WOVEN BY TOYOTA	USA
Freek Wiedijk	Plum Hall	USA
DAN PLAKOSH	CMU/SEI	USA
Henry Kleynhans	Bloomberg L.P.	USA
CHRISTOPHER BAZLEY	ARM	UK
MARTIN UECKER	TU GRAZ	Austria
JOSEPH MYERS	Red Hat	UK
JAKUB ŁUKASIEWICZ	MOTOROLA	PL
JeanHeyd Meneide		NL/NEN
Ori Bernstein		CA