731 2020-02 Prague ISO C++ Committee Trip Report — 🎉 C++20 is Done! 🎉 (self.cpp)

submitted 1 month ago * by blelbach NVIDIA | ISO C++ Library Incubator Chair | ISO C++ Tooling Chair

A very special video report from Prague.

C++20, the most impactful revision of C++ in a decade, is done! 🎉🎊🥳

At the ISO C++ Committee meeting in Prague, hosted by Avast, we completed the C++20 Committee Draft and voted to send the Draft International Standard (DIS) out for final approval and publication. Procedurally, it's possible that the DIS could be rejected, but due to our procedures and process, it's very unlikely to happen. This means that C++20 is complete, and in a few months the standard will be published.

During this meeting, we also adopted a plan for C++23, which includes prioritizing a modular standard library, library support for coroutines, executors, and networking.

A big thanks to everyone who made C++20 happen - the proposal authors, the minute takers, the implementers, and everyone else involved!

This was the largest C++ committee meeting ever - 252 people attended! Our generous host, Avast, did an amazing job hosting the meeting and also organized a lovely evening event for everyone attending.

This week, we made the following changes and additions to the C++20 draft:

- Improved the context-sensitive recognition of 'module' and 'import' to make it easier for non-compiler tools such as build systems to determine build dependencies.
- Added several new rangified algorithms.
- Added `ranges::ssize`.

cpp

join 126,704 readers

2,788 users here now

Discussions, articles, and news about the C++ programming language or programming in C++.

For C++ questions, answers, help, and advice see r/cpp_questions or StackOverflow.

- Refined the meaning of 'static' and 'inline' in module interfaces (P1779 and P1815).
- Resolved a lot of open core language and library issues and made many substantial improvements to specification.

The following notable features are in C++20:

- **Modules**.
- **Coroutines**.
- **Concepts**.
- **Ranges**.
- `constexpr` **ification:** `constinit`, `consteval`, `std::is_constant_evaluated`, `constexpr` **allocation**, `constexpr` `std::vector`, `constexpr` `std::string`, `constexpr` `union`, `constexpr` `try` **and** `catch`, `constexpr` `dynamic_cast` **and** `typeid`.
- `std::format("For C++{}", 20)`.
- `operator<=>`.
- Feature test macros.
- `std::span`.
- Synchronized output.
- `std::source_location`.
- `std::atomic_ref`.
- `std::atomic::wait`, `std::atomic::notify`, `std::latch`, `std::barrier`, `std::counting_semaphore`, etc.
- `std::jthread` and `std::stop_*`.

ABI Discussion

We had a very important discussion about ABI stability and the priorities of C++ this week in a joint session of the Language Evolution and Library Evolution group.

Although there was strong interest in exploring how to evolve ABI in the future, we are not pursuing making C++23 a clean ABI breaking release at this time. We did, however, affirm that authors should be encouraged to bring individual papers for consideration, even if those would be an ABI break. Many in the committee are interested in considering targeted ABI breaks when that would signify significant performance gains.

Get Started

The C++ Standard Home has a nice getting started page.

Videos

The C++ standard committee's education study group has a nice list of recommended videos.

Reference

cppreference.com

Books

There is a useful list of books on Stack Overflow. In most cases reading a book is the best way to learn C++.

Show all links

Filter out CppCon links

Show only CppCon links

a community for 11 years

MODERATORS message the moderators

IAmBabau
Verroq
pointfree
rectal_smasher_2000
STL MSVC STL Dev
cleroth Game Developer
blelbach
NVIDIA | ISO C++ Library Incubator Chair | ISO C++ Tooling Chair
BotTerminator

about moderation team »

> "How many C++ developers does it take to change a lightbulb?" — @tvaneerd
>
> "None: changing the light bulb is an ABI break." — @LouisDionne

Language Progress

Evolution Working Group Incubator (EWGI) Progress

The EWG Incubator met for three days in Prague and looked at and gave feedback to 22 papers for C++23. 10 of those papers were forwarded to Evolution, possibly with some revisions requested. Notably:

- Guaranteed copy elision for named return objects
- Generalized pack declaration and usage
- Member templates for local classes
- Object relocation in terms of move plus destroy

Several papers received a lot of feedback and will return to the Incubator, hopefully in Varna:

- A pipeline-rewrite operator
- Universal template parameters
- Partially mutable lambda captures
- C++ should support just-in-time compilation
- `move = bitcopies`

Notably, the proposed epochs language facility received no consensus to proceed. One significant problem pointed out was that in a concepts and modules world, we really cannot make any language changes that may change the satisfaction of a concept for a set of types. If one TU thinks `C<T>` is true, but another TU in a later epoch thinks `C<T>` is false, that easily leads to ODR violations. Many of the suggested changes in the paper run afoul of this problem. However, we're interested in solving the problem, so welcome an alternative approach.

Evolution Working Group (EWG) Progress

The top priority of EWG was again fixing the final national body comments for C++20. Once that was done, we started looking at C++23 papers. We saw a total of 36 papers.

Papers of note:

- We adopted the C++ IS schedule.
- We adopted a plan for C++23.

- We adopted a process for evolutionary proposals, to make sure that we reduce the chance that we'll make mistakes
- We agreed to pursue the Undefined Behavior group's effort to document Core Undefined or Unspecified Behavior going forward. They're documenting all language undefined behavior that C++ contains today, and we agreed to document and justify any new language undefined behavior going forward.

We marked 3 papers as tentatively ready for C++23:

- Make declaration order layout mandated
- Guaranteed copy elision for named return objects
- C++ Identifier Syntax using Unicode Standard Annex 31

They'll proceed to the Core language group at the next meeting if no issues are raised with these papers.

We continued reviewing pattern matching. This is one of our top priorities going forward. It's looking better and better as we explore the design space and figure out how all the corner cases should work. One large discussion point at the moment is what happens when no match occurs, and whether we should mandate exhaustiveness. There's exploration around the expression versus statement form. We're looking for implementation experience to prove the design.

We really liked deducing `this`, a proposal that eliminates the boilerplate associated with having `const` and non-`const`, `&` and `&&` member function overloads. It still needs wording and implementation experience, but has strong support.

We continue discussing floating-point fixed-layout types and extended floating point types, which are mandating IEEE 754 support for the new C++ `float16_t`, `float32_t`, `float64_t`, and adding support for `bfloat16_t`.

`std::embed`, which allows embedding strings from files, is making good progress.

In collaboration with the Unicode group, named universal character escapes got strong support.

`if consteval` was reviewed. We're not sure this is exactly the right solution, but we're interested in solving problems in this general area.

We saw a really cute paper on deleting variable templates and decided to expand its scope such that more things can be marked as `= delete` in the language. This will make C++ much more regular, and reduce the need for expert-only solutions to tricky problems.

Core Working Group (CWG) Progress

The top priority of CWG was finishing processing national body comments for C++20. CWG spent most of its remaining time this week working through papers and issues improving the detailed specification for new C++20 features.

We finished reviewing four papers that fine-tune the semantics of modules:

- We clarified the meaning of `static` (and unnamed namespaces) in module interfaces: such entities are now kept internal and cannot be exposed in the interface / ABI of the module. In non-modules compilations, we deprecated cases where internal-linkage entities are used from external-linkage entities. (These cases typically lead to violations of the One Definition Rule.)
- We clarified the meaning of `inline` in module interfaces: the intent is that bodies of functions that are not explicitly declared `inline` are not part of the ABI of a module, even if those function bodies appear in the module interface. In order to give module authors more control over their ABI, member functions defined in class bodies in module interfaces are no longer implicitly `inline`.
- We tweaked the context-sensitive recognition of the `module` and `import` keyword in order to avoid changing the meaning of more existing code that uses these identifiers, and to make it more straightforward for a scanning tool to recognize these declarations without full preprocessing.
- We improved backwards compatibility with unnamed enumerations in legacy header files (particularly C header files). Such unnamed enumerations will now be properly merged across header files if they're reachable in multiple different ways via imports.
- We finalized some subtle rules for concepts: a syntax gotcha in `requires` expressions was fixed, and we allowed caching of concept values,

which has been shown to dramatically improve performance in some cases.

- We agreed to (retroactively, via the defect report process) treat initialization of a `bool` from a pointer as narrowing, improving language safety.
- We added permission for a comparison function to be defaulted outside its class, so long as the comparison function is a member or friend of the class, for consistency and to allow a defaulted comparison function to be non-inline.

Library Progress

Library Evolution Working Group Incubator (LEWGI) Progress

LEWGI met for three and a half days this week and reviewed 22 papers. Most of our work this week was on various numerics proposals during joint sessions with the Numerics group. A lot of this work may end up going into the proposed Numerics Technical Specification, whose scope and goals we are working to define. We also spent a chunk of time working on modern I/O and concurrent data structures for the upcoming Concurrency Technical Specification Version 2.

LEWGI looked at the following proposals, among others:

- Numerics:
  - Physical Units Library.
  - Linear Algebra.
- Concurrency:
  - Concurrent Queues.
- Low-level File I/O
  - Mapped File Handle.
- Narrowing Conversions
  - std::is_narrowing_conversion.
  - std::narrow_cast.
- Random Numbers
  - Improving std::random_device.
  - Portable Distributions.
  - Improving Engine Seeding.

Library Evolution Working Group (LEWG) Progress

After handling the few remaining National Body comments to fix issues with C++20, LEWG focused on making general policy decisions about standard library design standards. For example, we formally codified the guidelines for concept names in the standard library, and clarified SD-8, our document listing the compatibility guarantees we make to our users. Then we started looking at C++23 library proposals.

Moved-from objects need not be valid generated much internal discussion in the weeks leading up to the meeting as well as at the meeting itself. While the exact solution outlined in the paper wasn't adopted, we are tightening up the wording around algorithms on what operations are performed on objects that are temporarily put in the moved-from state during the execution of an algorithm.

The biggest C++23 news: LEWG spent an entire day with the concurrency experts of SG1 to review the **executors** proposal — we liked the direction! This is a huge step, which will enable **networking**, **audio**, **coroutine library support**, and more.

Other C++23 proposals reviewed include

- a new `status_code` facility
- an ability for containers and allocators to communicate about the actual allocation size
- iterator range constructors for `std::stack` and `std::queue`

We've also decided to deprecate `std::string`'s assignment operator taking a `char` (pending LWG).

Library Working Group (LWG) Progress

The primary goals were to finish processing NB comments and to rebase the Library Fundamentals TS on C++20. We met both of those goals.

We looked at all 48 open library-related NB comments and responded to them. Some were accepted for C++20. Some were accepted for C++20 with changes. For some, we agreed with the problem but considered the fix to be too risky for C++20, so an issue was opened for consideration in C++23. For many the response was "No consensus for change," which can mean a variety of things from “this is not really a problem” to “the problem is not worth fixing.”

The last of the mandating papers was reviewed and approved. All of the standard library should now be cleaned up to use the latest library wording guidelines, such as using “Mandates” and “Constraints” clauses rather than “Requires” clauses.

Some time was spent going through the LWG open issues list. We dealt with all open P1 issues (“must fix for C++20”). Many of the open P2 issues related to new C++20 features were dealt with, in an attempt to fix bugs before we ship them.

This was Marshall Clow’s last meeting as LWG chair. He received a standing ovation in plenary.

Concurrency and Parallelism Study Group (SG1) Progress

SG1 focused on C++23 this week, primarily on driving executors, one of the major planned features on our roadmap. Executors is a foundational technology that we'll build all sorts of modern asynchronous facilities on top of, so it's important that we land it in the standard early in the C++23 cycle.

At this meeting, LEWG approved of the executors design, and asked the authors to return with a full specification and wording for review at the next meeting.

SG1 reviewed and approved of a refinement to the design of the sender/receiver concepts. This change unifies the lifetime model of coroutines and sender/receiver and allows us to statically eliminate the need for heap allocations for many kinds of async algorithms.

Going forward, SG1 will start working on proposals that build on top of executors, such as concurrent algorithms, parallel algorithms work, networking, asynchronous I/O, etc.

Networking Study Group (SG4) Progress

SG4 started processing review feedback on the networking TS aimed at modernizing it for inclusion in C++23. SG4 also reviewed a proposal to unify low-level I/O with the high-level asynchronous abstractions and gave feedback to the author.

Numerics Study Group (SG6) Progress

The Numerics group met on Monday this week, and also jointly with LEWGI on Tuesday and Thursday, and with SG19 on Friday.

We reviewed papers on a number of topics, including:

- The scope and goals for the Numerics TS.
- Linear algebra.
- Units library.

Compile-Time Programming Study Group (SG7) Progress

Circle is a fork of C++ that enables arbitrary compile-time execution (e.g. a compile-time `std::cout`), coupled with reflection to allow powerful meta-programming. SG7 was interested in it and considered copying parts of it. However, concerns were raised about security and usability problems, so the ability to execute arbitrary code at compile-time was rejected.

Besides that, we also continued to make progress on C++ reflection including naming of reflection keywords and potential to enable lazy evaluation of function arguments.

We also looked at the JIT proposal and asked authors to try to unify the design with current reflection proposals.

Undefined Behavior Study Group (SG12)/Vulnerabilities Working Group (WG23) Progress

We set out to enumerate all undefined and unspecified behavior. We've decided that upcoming papers adding new undefined or unspecified behavior need to include rationale and examples.

SG12 also collaborated with the MISRA standard for coding standards in embedded systems to help them update the guidelines for newer C++ revisions.

Human Machine Interface and Input/Output Study Group (SG13) Progress

SG13 had a brief presentation of extracts from the 2019 CppCon keynote featuring Ben Smith (from 1:05:00)

We looked at A Brief 2D Graphics Review and encouraged exploration of work towards a separable color proposal.

Finally, we worked through the use cases in Audio I/O Software Use Cases. We have a couple of weeks before the post meeting mailing deadline to collect additional use cases and will then solicit feedback on them from WG21 and the wider C++ community.

Tooling Study Group (SG15) Progress

The Tooling study group met this week to continue work on the Module Ecosystem Technical Report. Three of the papers targeting the Technical Report are fairly mature at this point, so we've directed the authors of those papers to work together to create an initial draft of the Technical Report for the Varna meeting. Those papers are:

- Dependency Information Format
- Module Recipe and BMI Reuse
- User-Facing Lexicon and File Extensions

This draft will give us a shared vehicle to start hammering out the details of the Technical Report, and a target for people to write papers against.

We also discussed two proposals, about debugging C++ coroutines and asynchronous call stacks.

Machine Learning Study Group (SG19) Progress

SG14 met in Prague in a joint session with SG19 (Machine Learning).

The freestanding library took a few steps forward, with some interesting proposals, including Freestanding Language: Optional `::operator new`

One of the biggest decisions was on Low-Cost Deterministic C++ Exceptions for Embedded Systems which got great reactions. We will probably hear more about it!

Unicode and Text Study Group (SG16) Progress

Our most interesting topic of the week concerned the interaction of execution character set and compile-time programming. Proposed features for `std::embed` and reflection require the evaluation of strings at compile time and this occurs at translation phase 7. This is after translation phase 5 in which character and string literals are converted to the execution character set. These features require interaction with file names or the internal symbol table of a compiler. In cross compilation scenarios in which the target execution character set is not compatible with the compiler's host system or internal encoding, interesting things happen. As in so many other cases, we found an answer in UTF-8 and will be recommending that these facilities operate solely in UTF-8.

We forwarded Named Universal Character Escapes and C++ Identifier Syntax using Unicode Standard Annex 31 to EWG. Both papers were seen by EWG this week and are on track for approval for C++23 in meetings later this year.

We forwarded Naming Text Encodings to Demystify Them to LEWG.

We declined to forward a paper to enhance `std::regex` to better support Unicode due to severe ABI restrictions; the `std::regex` design exposes many internal details of the implementation to the ABI and implementers indicated that they cannot make any significant changes. Given the current state of `std::regex` is such that we cannot fix either its interface or its well-known performance issues, a number of volunteers agreed to bring a paper to deprecate `std::regex` at a future meeting.

Machine Learning Study Group (SG19) Progress

SG19 met for a full day, one half day with SG14 (Low Latency), and one half day with SG6 (Numerics).

Significant feedback from a ML perspective was provided on Simple Statistics functions, especially regarding the handling of missing data, non-numeric data, and various potential performance issues.

There was an excellent presentation of "Review of P1708: Simple Statistical Functions" which presented an analysis across Python, R, SAS and Matlab for common statistical methods.

The graph library paper had a great reaction, was also discussed, and will proceed.

Also, support for differentiable programming in C++, important for well-integrated support for ML back-propagation, was discussed in the context of differentiable programming for C++.

Contracts Study Group (SG21) Progress

In a half-day session, we discussed one of the major points of contention from previous proposals, which was the relationship between “assume” and “assert”, disentangling colloquial and technical interpretations. We also discussed when one implies the other, and which combinations a future facility should support.

- Previous Disagreements
- Assumptions
- Portable Assumptions

C++ Release Schedule

NOTE: This is a plan not a promise. Treat it as speculative and tentative. See P1000 for the latest plan.

- IS = International Standard. The C++ programming language. C++11, C++14, C++17, etc.
- TS = Technical Specification. "Feature branches" available on some but not all implementations. Coroutines TS v1, Modules TS v1, etc.
- CD = Committee Draft. A draft of an IS/TS that is sent out to national standards bodies for review and feedback ("beta testing").

Meeting	Location	Objective
~~2018 Summer LWG Meeting~~	~~Chicago~~	~~Work on wording for C++20 features.~~
~~2018 Fall EWG Modules Meeting~~	~~Seattle~~	~~Design modules for C++20.~~
~~2018 Fall LEWG/SG1 Executors Meeting~~	~~Seattle~~	~~Design executors for C++20.~~

Meeting	Location	Objective
~~2018 Fall Meeting~~	~~San Diego~~	~~C++20 major language feature freeze.~~
~~2019 Spring Meeting~~	~~Kona~~	~~C++20 feature freeze. C++20 design is feature-complete.~~
~~2019 Summer Meeting~~	~~Cologne~~	~~Complete C++20 CD wording. Start C++20 CD balloting ("beta testing").~~
~~2019 Fall Meeting~~	~~Belfast~~	~~C++20 CD ballot comment resolution ("bug fixes").~~
2020 Spring Meeting	**Prague**	**C++20 CD ballot comment resolution ("bug fixes"), C++20 completed.**
2020 Summer Meeting	Varna	First meeting of C++23.
2020 Fall Meeting	New York	Design major C++23 features.
2021 Winter Meeting	Kona	Design major C++23 features.
2021 Summer Meeting	Montréal	Design major C++23 features.
2021 Fall Meeting	🗺️	C++23 major language feature freeze.
2022 Spring Meeting	Portland	C++23 feature freeze. C++23 design is feature-complete.
2022 Summer Meeting	🗺️	Complete C++23 CD wording. Start C++23 CD balloting ("beta testing").
2022 Fall Meeting	🗺️	C++23 CD ballot comment resolution ("bug fixes").
2023 Spring Meeting	🗺️	C++23 CD ballot comment resolution ("bug fixes"), C++23 completed.

Meeting	Location	Objective
2023 Summer Meeting	🗺️	First meeting of C++26.

Status of Major C++ Feature Development

NOTE: This is a plan not a promise. Treat it as speculative and tentative.

- IS = International Standard. The C++ programming language. C++11, C++14, C++17, etc.
- TS = Technical Specification. "Feature branches" available on some but not all implementations. Coroutines TS v1, Modules TS v1, etc.
- CD = Committee Draft. A draft of an IS/TS that is sent out to national standards bodies for review and feedback ("beta testing").

Changes since last meeting are in bold.

Feature	Status	Depends On	Current Target (Conservati Estimate)
Concepts	Concepts TS v1 published and merged into C++20		C++20
Ranges	Ranges TS v1 published and merged into C++20	Concepts	C++20
Modules	Merged design approved for C++20		C++20
Coroutines	Coroutines TS v1 published and merged into C++20		C++20

Feature	Status	Depends On	Current Target (Conservati Estimate)
Executors	New compromise design approved for C++23		C++26
Contracts	Moved to Study Group		C++26
Networking	Networking TS v1 published	Executors	C++26
Reflection	Reflection TS v1 published		C++26
Pattern Matching			C++26
Modularized Standard Library			**C++23**

Last Meeting's Reddit Trip Report.

If you have any questions, ask them in this thread!

Report issues by replying to the top-level stickied comment for issue reporting.

/u/blelbach, Tooling (SG15) Chair, Library Evolution Incubator (SG18) Chair

/u/bigcheesegs

/u/c0r3ntin

/u/jfbastien, Evolution (EWG) Chair

/u/arkethos (aka code_report)

/u/vulder

/u/hanickadot, Compile-Time Programming (SG7) Chair

/u/tahonermann, Text and Unicode (SG16) Chair

/u/cjdb-ns, Education (SG20) Lieutenant

/u/nliber

/u/sphere991

/u/tituswinters, Library Evolution (LEWG) Chair

/u/HalFinkel, US National Body (PL22.16) Vice Chair

/u/ErichKeane, Evolution Incubator (SG17) Assistant Chair

/u/sempuki

/u/ckennelly

/u/mathstuf

/u/david-stone, Modules (SG2) Chair and Evolution (EWG) Vice Chair

/u/je4d, Networking (SG4) Chair

/u/FabioFracassi, German National Body Chair

/u/redbeard0531

/u/nliber

/u/foonathan

/u/InbalL, Israel National Body Chair

/u/zygoloid, C++ Project Editor

⋯ and others ⋯

514 comments share save hide report

top 200 comments show 500

sorted by: **best**

Want to add to the discussion?

Post a comment!

CREATE AN ACCOUNT

[–] **blelbach** NVIDIA | ISO C++ Library Incubator Chair | ISO C++ Tooling Chair [S,M] [score hidden] 28 days ago - stickied comment

This a top-level stickied comment for reporting issues in the trip report.

permalink embed save report reply

[–] **James20k** P2005R0 80 points 1 month ago

This was my first committee meeting! It was extremely interesting, it answered a lot of my questions about why C++ has gotten to the state which it is in, in both the good and the bad. Apparently I've now become the colour guy which is nice too

If you've got any questions about the process I can answer them as best I can, I mostly hung out in LEWGI looking at library proposals, though i jumped around a lot (as well as presenting to SG13 about the graphics proposal), and was there for the great ABI bakeoff

I think a few things are worth saying though

1. Everyone was extremely friendly. Thank herb for this, as its been a big goal of his
2. The committee has a lack of technically expert manpower in many fields. If you work for gamedev, or know a lot about clang/gcc/msvc/icc, or know a lot about the language you should really go because it

needs you folks. I floated a few times my idea that we should always have an implementer on a phone hotline, but it costs £10 every time you phone it

3. Everyone in the committee is painfully aware of the language problems. Its not lack of enthusiasm or acknowledgement that means stuff isn't being fixed, although in some cases (eg random), there is a lack of domain expertise that means that a subgroup might not really understand that an issue is so important (eg uniform_etc_distribution)

I believe you're allowed to publicly share straw polls, but not directly quote anyone without permission, though I'd love to know more exactly what the rules are around sharing eg "x group thought y"

Oh and please *please* go if you're gamedev. There were 6 of us there in total. Often i was the sole voice of game development in the room, which is slightly disconcerting

permalink embed save report reply

[–] **adnukator** 27 points 1 month ago*

It was my first committee meeting as well and I fully agree with the above. Too bad I had other obligations and could stay only for two days. Might change with the next meeting in Varna.

The friendliness was really refreshing, considering the amount of pitchforks and torches any slightly controversial statement on the internet can summon. The whole meeting made me seriously consider writing and submitting an idea I've been keeping in my head for a while. Until now I had the assumption that any proposal going before the committee has to be bullet-proof. But it actually turns out that if your proposal does have flaws, nobody gangs up on you. Instead, you get constructive feedback on what to improve and get suggestions which paths to further explore to arrive at a more polished proposal.

I'd seriously recommend visiting a C++ committee meeting to anyone who either wants to improve the language or just wants to learn how things are made. Even to any C++ haters - and perhaps "convert" them in the process. I felt that valid remarks during debates are welcome from anyone. So the more people with different specializations chime in (productively, of course), the better the proposals can become.

permalink embed save parent report reply

[–] **variar_fav** 15 points 1 month ago

My first meeting also. +1 for earlier comments. I didn't have a proposal for C++, my goal was to visit different rooms and get better understanding of how baking C++ works. This is very interesting and unusual process. Each decision has to be considered in terms of what impact it will have during next decades. Or for example will it provoke people to use marcos (that many hate but sometimes language leaves no choice). Each study group feels different, some are easier than others. Didn't have guts to visit CWG or LWG ;) (however I understand that I couldn't do anything useful there this time).

After the meeting I feel encouraged to convert some ideas into P-papers. That really is not very scary.

I want to thank group chairs. These folks did a lot of work to make discussions go smoothly and progress. Can't imagine how much they had to do behind the scene to manage schedules, meeting notes and polls, getting needed people in the room (eg. implementors) etc.

If a C++ committee meeting is happening somewhere near you, you should consider visiting it. One don't have to write a proposal, being able to share real life field experience with using C++ is already big help. However, it's worth reading paper proposals in advance :)

permalink embed save parent report reply

[–] **imgarfield** 31 points 1 month ago

The lack of gamedev involvement is extremely ironic and unfortunate, considering they are arguably one of the biggest C++ user and *the* biggest in terms of mainstream employment.

permalink embed save parent report reply

[–] **James20k** P2005R0 17 points 1 month ago

Yep! Its crazy there's no gamedev. The topic of the lack of portability of random number distributions cropped up, which is one of the major reasons why gamedev would never use them. If there'd been a big gamedev presence in the room, we could have gotten it through, but as it was most of the represented industry in the room didn't care, so it died

permalink embed save parent report reply

[–] **SeanMiddleditch** Game Developer 16 points 1 month ago

This was a huge reason why we started SG14, though it eventually morphed into low-latency rather than being purely about games.

We used to hold SG14 meetings at GDC and such even (not sure if that has happened recently; I'm not involved anymore).

permalink embed save parent report reply

[–] **James20k** P2005R0 4 points 1 month ago

Interesting, I had no idea. Thanks!

permalink embed save parent report reply

[–] **SAHChandler** C++ Bruja 25 points 1 month ago*

a lot of times on twitter you'll see gamedevs say one of

1) the committee is full of academic masturbation! why would I go no one would listen to me? 2) the committee should come to us, because that's how publishers work and that's what we're used to 3) they just need to focus on this problem *I* have and nothing else 4) they should just make C++ more like C#

I think people change their tune when they do finally go but overall, and this is gonna sound petty, (   ) I feel like a lot of the games industry is full of pillow princesses who don't understand that they need to take an initiative because the world really doesn't care about them `¯\_(ツ)_/¯`

That said I try to keep gamedev needs in mind because I used to work in that field and have a ton of contacts and friends there, but sometimes the requests I hear from people usually boil down to "Microsoft had this issue with their compiler (in 2003) so obviously the whole language works like that and nothing has changed" :/

permalink embed save parent report reply

[–] **--Jasper--** 11 points 1 month ago

The big reason is that this takes time. A lot of time. We're focused on shipping games most of the day, and we have a lot of stuff to get through. Hal Finkel admitted writing papers takes, at minimum, a month or so of effort. That's a super high bar for us, and that's time we don't have.

If the process was more open and allowed us to even *comment* on papers going through the process without having to write a replacement, we'd love that.

I'm not a big fan of the rough "us vs. them" characterization, re: pillow princesses or whatever. We work with 20-year-old engine code that barely has any tests, come on, we're as jaded as the rest of them. On our side of the fence, we look at "std::byte" and think you're all pillow princesses playing with ivory dollhouses too. :)

As-is though, the process is open to "those who have time to write a paper", and that's not something gamedev can afford to fund right now.

permalink embed save parent report reply

[–] **SAHChandler** C++ Bruja 14 points 1 month ago

Most of the people on the committee (save for implementers who typically wear several hats at their various orgs/work on multiple products and Dr. Walter E. Brown who is retired and been given emeritus status due to his massive list of contributions) are also focused on shipping and maintaining their products. When I worked at Apple we had to ship a deployment every 3 weeks for software that (at the time) received 8 billion requests a day from Japan alone. It's only gotten bigger since.

As for commenting on papers, you're more than free to email each author (that's why the Reply-To field is placed in each paper). It also only costs 2200USD to join the American National Body (INCITS, not ANSI). If an industry where the CEOs of Rockstar, Activision, and Epic Games are walking away with billions of dollars from GTA, Modern Warfare, and Fortnite, while developers get left out in the cold, maybe something needs to change. The process is (mostly) closed because, unfortunately, collusion is a concern for multiple governments and ISO provides protections for its various members. Imagine for a moment if GOG, Tim Sweeney, and Gabe Newell sat in a room together to discuss a "game store standard" for PC. If it wasn't done via a Business League or something similar to ISO, the FBI (or any other nation state's police force) could walk into the room and arrest them.

The reason std::byte exists is because `std::is_same_v<char, signed char>` is false and the signedness of char is compiler flag specific. If it's a signed char (which again is not the same as `signed char` because of C), then overflow is implementation defined. If it's unsigned, then it will behave correctly and this can affect codegen. std::byte is the *only* way to

1. make writing "unsigned char" not a PITA
2. Remove the ability to perform mathematical operations on a byte (i.e., you can only do bitwise operations)
3. Keep strict aliasing as a compiler fence without requiring volatile reads/writes for no reason other than "I want to enforce strict aliasing"

Lastly, pillow princess is a term from the LGBTQ+ community that implies someone wants to be "serviced" and not have to do anything. In other words, they can just lie back on a pillow and... well you get the idea :)

But to get access to the mailing lists it's the cost of rent for a 1b1b apartment in Berkeley and that's a business expense. If game companies that make as much money as Epic Games does feel like they can't participate, that's on them. Plenty of small companies (smaller than many indie studios, even) participate in and show up to these meetings. There's no valid excuse coming from game companies.

permalink embed save parent report reply

[–] **m_ninepoints** rendering/gfx, game engines 8 points 1 month ago

I'm in the games industry currently and am somewhat sympathetic to both sides. First, I should say that while there is some money in games, it is absolutely dwarfed by the likes of Google, Apple, Microsoft, etc due to production costs. Hundreds of millions of dollars of revenue per title isn't actually all that much when each title costs up to 100M + marketing costs to ship (restricting myself to AAA here).

I've actually sent a lot of notes regarding, say, the graphics proposal before and why I'm pretty much vehemently against what was outlined, although James did a much better job formalizing the argument then I did.

To clear the air a bit, I think it isn't so much that game developers want free "service." It's more that they don't like abstractions added that hurt compile times without clear performance gains or advantages. Remember that in most studios, the STL is nowhere to be found because a lot of this code predated the existence of a move constructor. Move constructors certainly improved things a bit, but until trivially relocatable traits are in, the code provided by the STL is *still* slower than what many of us use in house. Even Google et. al. uses their own bespoke "standard" library.

When you couple crunch and stress, plus a lot of baggage and getting marginally less pay coding something because you're in a more glamorous field, it tends to turn into an unproductive whine. To your point, this irks me too, and I wish that the criticisms levied by others in my industry could be done more constructively.

As for companies like Epic, I see UE4 code all the time. This code is NOT idiomatic "modern" C++ by any stretch. In fact, the abstractions of modern C++ have left the game dev world in many places, so for most of them, they would almost prefer it if the C++ language ossified completely and all remaining improvements were made just to the tooling.

Personally, I really wish the generalizations would stop on both sides. Pointing a finger at all game developers is just as egregious as pointing a finger at all committee members. It places a disproportionate significance on the voice of the vocal minority, when some of us actually like some of the features coming down the pipeline, and would probably even sit down to write a paper if we didn't have so many other things going on.

permalink embed save parent report reply

[–] **kmhofmann** https://selene.dev 3 points 29 days ago

> In fact, the abstractions of modern C++ have left the game dev world in many places, so for most of them, they would almost prefer it if the C++ language ossified completely and all remaining improvements were made just to the tooling.

I also dislike generalizations, but you're making one here, and... it's kind of true!

In fact, this is my single biggest criticism targeted at many C++ developers in the games industry. They seem to hate progress and rile against it loudly on Twitter. Something I will never understand, since a lot of the beauty of C++ is in its very powerful abstractions.

Yup, this was a massive generalization, but one unfortunately I see confirmed over and over again.

permalink embed save parent report reply

load more comments (2 replies)

load more comments (1 reply)

load more comments (3 replies)

load more comments (1 reply)

[–] **14ned** LLFIO & Outcome author | Committees WG21 & WG14 7 points 1 month ago

Lot of people on the current committee find the current situation with standard RNGs unfortunate. However, we were all in other rooms proposing other stuff, so we weren't in that room.

For the record, there's easily half a dozen, perhaps a dozen game devs attending WG21. I'd even say they have outsize weight influence for their number. But all were elsewhere working on higher priority items this week. Sorry.

permalink embed save parent report reply

[–] **James20k** P2005R0 3 points 1 month ago

No need to apologise! I was in the room, and in hindsight knowing what I know now, I'd speak up next time and make a much louder noise. It was partly a learning experience for me to realise that even if something seems obvious, you need to really make the case

But yeah I mean, there were a few other things on the docket so not exactly surprising folks were busy!

permalink embed save parent report reply

[–] **14ned** LLFIO & Outcome author | Committees WG21 & WG14 3 points 1 month ago

I'll also be honest and say that I *personally* didn't find the RNG proposals as presented compelling. Not due to their goal, but rather the formulation proposed. If others felt the same way as me, that might also explain the absence of support.

I have half a mind to propose a `file_handle` implementation which implements a synthesised seekable file full of random data. I use an implementation for testing and benchmarking in LLFIO. It's identical on all platforms, vectorises with buffers filled, has an ungodly performance. The sequence can be replayed by reusing the same offset read from, and the "contents" can be permuted via setting a seed.

I find it a bit ugly to solve the RNG problem in standard C++, but it very definitely solves all *my* problems with RNG in standard C++.

permalink embed save parent report reply

[–] **James20k** P2005R0 3 points 1 month ago

Yeah, the proposals weren't well motivated in that presentation, and the author did not do the best job at selling it. There's an alternate formulation there that needs to be done and re-presented

Hah. I mean its honestly not the worst method for doing rng. Ill take anything over rand()

permalink embed save parent report reply

[–] **Dascandy** HippoMocks/cpp-dependencies/Evoke/Pixel dev 2 points 1 month ago

+1 - worked with Martin on that paper (not listed I think) and I'm 100% in favor of making the distributions portable - yet I wasn't in that room, because there were other rooms I had to be in more.

permalink embed save parent report reply

[–] **Benjamin1304** 6 points 1 month ago

I find it very sad that the committee members seem to only care about the persons physically present in the room. It's quite easy to understand why the non-portability of random number distributions is a problem for game devs, probably one of the biggest C++ community out there btw, no matter if there is only one of them in the room raising the issue.

I really think that the standardization process should happen more online where it's easy to reach for the community rather basing all decisions on the couple hundred people having the ability to travel to the meetings.

permalink embed save parent report reply

[–] **James20k** P2005R0 8 points 1 month ago

Its not that... Committee members don't care. Its that every feature in C++ has a significant time investment to solve, so authors proposing features have to justify them. If the rest of the room doesn't think its that important, it'll get voted against. A strong case was made that non portability doesn't matter most of the time, and nobody managed to make a strong enough case that it does

Its not that the committee members ignored anyone, its just that the people on the other side of the argument weren't there to make it, and committee members only know their own domains

It was actually very early in the week, if it'd been later I'd have made a case for it knowing now how the process works if the author doesn't manage to convince the room - but I believe it was monday or tuesday and I was more green

permalink embed save parent report reply

[–] **Dascandy** HippoMocks/cpp-dependencies/Evoke/Pixel dev 8 points 1 month ago

There are at least 5 different disjoint groups of people that all think they're the biggest group of C++ users.

permalink embed save parent report reply

load more comments (7 replies)

load more comments (5 replies)

load more comments (6 replies)

[–] **Ameisen** 10 points 1 month ago

I'd love to go. Can't afford to, though.

permalink embed save parent report reply

[–] **James20k** P2005R0 11 points 1 month ago

Write a paper and get funded to go! That's how I managed to afford going

They'll pay for travel and accommodation, though not food

permalink embed save parent report reply

[–] **bumblebritches57** FoundationIO, OVIA, and occasionally LLVM 3 points 1 month ago

Hmm, how does that work?

I'm thinking about writing a proposal for WG14/C but my main concern was not being able to go in person.

permalink embed save parent report reply

[–] **James20k** P2005R0 10 points 1 month ago

I'm a brit, I'll quickly describe the process

1. Wrote a paper
2. People liked the paper
3. Got contacted by the head of SG13 (Roger Orr) asking me if I wanted to present
4. Sent the head of the BSI (also Roger Orr) an email asking for funding, who asked me to forward to herb
5. Simultaneously someone asked me to come along to the BSI, the british national body for C++
6. Got on well at the BSI
7. Turns out my email to Herb got lost, and Roger Orr poked him in his capacity as BSI head (I think? its not exactly overly formal)
8. Herb said yes, in conjunction with some other people

Now I have to send them receipts for stuff

If you need help with the process involved in writing a proposal... I can't say I'm the best person in the entire universe to ask (only written 1 paper, and this was my first meeting), but I'm happy to help if I can or direct you to other people who know more. Its a lot easier once you've been to a meeting I think, because you know everyone then

If you need help getting funding just email someone and they'll be happy to help (or I can point you towards people)

permalink embed save parent report reply

[–] **FabioFracassi** C++ Committee | Consultant 6 points 1 month ago

Wg14/C is a totally different committee (apart from a few people who go to both) , with different rules (and afaiu much less open).

For wg21/c++ isocpp.org takes care of this.

permalink embed save parent report reply

[–] **14ned** LLFIO & Outcome author | Committees WG21 & WG14 5 points 1 month ago

WG14/C is indeed a totally different committee. Technically the same rules apply for both, due to ISO, but in practice different emphasis of rules has appeared.

WG14 is just as open as WG21. Perhaps even more so. They'll be delighted to see anybody turn up, and because it's small, it's intimate in a way WG21 once was, and no longer can be. You also get a mix of everybody in a single room, and that is vastly more efficient and productive than in WG21 where it can take several meetings before your proposal gets shot down.

Achieving anything but minor change at WG14 is very, very hard. They'll gladly hear you out, even very radical proposals, but unless it's correctness you're fixing, you'll probably be refused.

As WG14 control the C stuff, proposing changes to C stuff at WG21 will usually result in being told to go ask WG14.

WG21 and WG14 will intentionally colocate meetings some time in 2021, so if you attend then, you can go to both. There is no funding for attending WG14 meetings, but there is for WG21 meetings under some circumstances, so that could be a solution. Just time submitting your proposal right!

permalink embed save parent report reply

load more comments (3 replies)

[–] **Xeverous** Worker & Hobbyist with own C++ website under construction 3 points 1 month ago

> If you work for gamedev, or know a lot about clang/gcc/msvc/icc, or know a lot about the language you should really go because it needs you folks

I would like to go but any such meeting far away concerns me how it can break my job. Where do committee people work in? Are they purely working on C++, funded by their companies are the meetings just small gap in their job allowed by their employer? Money is not a problem for me I but have no idea how I can make a full-time job with far flies every few months.

permalink embed save parent report reply

[–] **STL** MSVC STL Dev 8 points 1 month ago

Being away from work for a week is indeed a significant cost aside from money. If your employer uses C++ to any significant degree, you should be able to argue that (1) you can represent your company's concerns as a user even if you aren't driving any proposals (being able to vote in straw polls is a big deal), and (2) attending a Committee meeting teaches you a lot about the latest and upcoming developments in the language - training which is hard to access anywhere else (typically books etc. cover a standard that's 3+ years old).

permalink embed save parent report reply

[–] **Xeverous** Worker & Hobbyist with own C++ website under construction 5 points 1 month ago

I currently work in outsourcing company so the technology choice is actually on the client. My company just coordinates hiring, training, PR/HR/integration and people based on their skills.

On the other hand, there was a recent post on company's blog that someone got sponsored from the "passion/hobby sponsoring program" and the post contents were about getting some Coroutine proposal thing done and the author got the travel/meeting cost sponsored. I guess I should contact that person and ask for the guidance... (don't know the author in person, very likely not working in the office/city as I)

permalink embed save parent report reply

[–] **Daniela-E** Modern C++, Embedded 4 points 1 month ago

During this week in Prague I happended to run into some other C++ devs from Germany who (like me) do this on their own initiative, out of their own money, spending some of their off-days budget. In other words: without any support or funding from the company they are employed at. In my particular case, I'm fully employed as a C++ dev in a tiny company developing and building highly customized industrial machines deployed to factory floors. This kind of business is really tough, getting any substantial support from a company like this is nearly impossible.

permalink embed save parent report reply

[–] **Rseding91** Factorio Developer 2 points 1 month ago

> If you work for gamedev

What exactly are they looking for from people in gamedev?

permalink embed save parent report reply

[–] **matthieum** 36 points 1 month ago

I am, perhaps unreasonably, very excited about the `move = bitcopies` proposal.

It is my personal opinion that C++ should aim for best-in-class performance. After all, performance is often the core reason for choosing to use C++, thus sub-par performance should be a significant worry for renewed usage.

This proposal addresses the core performance issue with move semantics as defined today, allowing significantly faster implementations. For example, suddenly growing a `std::vector<std::unique_ptr<T>>` can use `realloc`.

permalink embed save report reply

[–] **whichton** 4 points 1 month ago

That is a great and very necessary paper. I am still not clear what is the difference in objective between this paper and P1144: Object relocation in terms of move plus destroy and why /u/14ned wants to vacate the "relocation space". Both seem to achieve the same goal, albeit in different ways.

permalink embed save parent report reply

[–] **14ned** LLFIO & Outcome author | Committees WG21 & WG14 9 points 1 month ago

P1144 enables standard library containers to be less stupid with collections of some types. It does not modify ABI of such types otherwise e.g. return of them from functions.

P1029 is the opposite almost: types opted into move bitcopying get improved codegen i.e. ABI break over if they were not opted in.

Both proposals enable standard library containers to be less inefficient, however P1144 produces superior efficiency improvements to P1029 for standard library containers.

permalink embed save parent report reply

[–] **VisualSlice3** 3 points 1 month ago

I really like P1029 it seems quite simple for what it does.

If this was to get shipped do you think implementers would take the hit, break ABI and apply it to existing types like unique_ptr and friends?

permalink embed save parent report reply

[–] **14ned** LLFIO & Outcome author | Committees WG21 & WG14 2 points 1 month ago

I suspect that would be extremely unlikely for existing architectures, but highly likely for any newly supported architectures.

For specifically `unique_ptr`, I can see it becoming ABI broken if say a macro like `LIBCXX_ABI_UNSTABLE` were defined, or equivalent thereof for the various standard library implementations.

permalink embed save parent report reply

load more comments (2 replies)

[–] **smdowney** 24 points 1 month ago

It's my fault you can't `throw 💩;` anymore.

permalink embed save report reply

[–] **SAHChandler** C++ Bruja 5 points 1 month ago

😭

permalink embed save parent report reply

[–] **Dascandy** HippoMocks/cpp-dependencies/Evoke/Pixel dev 5 points 1 month ago

I argumented it such that it got zero votes against.

permalink embed save parent report reply

[–] **PeterBrett** SG16; CAD software dev 2 points 1 month ago

My hero ❤

permalink embed save parent report reply

[–] **c0r3ntin** 2 points 1 month ago

error: unexpected character U+2764 at line 1

permalink embed save parent report reply

[–] **HildartheDorf** 2 points 29 days ago

I thought valid unicode was acceptable for identifiers. Did you explicitly ban emoji in identifiers?

permalink embed save parent report reply

[–] **smdowney** 7 points 29 days ago

No, implicitly. We're fixing what's in allowed identifiers based on the Unicode TR31 standard. I want us to do a better job at supporting Unicode, that is supporting identifiers in all languages. That means not allowing things like arbitrary LTR modifiers, zero width spaces, punctuation, etc. The Unicode standard has tables of the characters that are good for identifiers. They don't include emoji. At least partly because you need, by design, left-to-right mods and zero width joiners, to express all emoji.

Someone could propose adding PILE OF POO to the list of allowed initial characters. We did that for LOW BAR (_) to match the current grammar.

permalink embed save parent report reply

[–] **AlexAlabuzhev** 21 points 1 month ago*

https://en.cppreference.com/w/cpp/chrono/duration:

> Literals
>
> h, min, s, ms, us, ns
>
> Note: the literal suffixes d and y do not refer to days and years but to day and year, respectively. (since C++20)

I.e. `1d` is not `24h`, but the 1st day of a month.

What's the motivation for this?

Yes, now we can construct a `year_month_day` as `15d/February/2020`, but is constructing dates *from literals in the code* (in 3 different ways) really something needed every day and important enough to justify *more* inconsistency?

It feels quite like `initializer_list` (convenient for helloworlding & unit tests, but rarely used in the actual code and breaks uniform initalization beyond repair).

permalink embed save report reply

[–] **tpecholt** 17 points 1 month ago

I never understood the push for overloaded / for date construction. Many countries including mine use different separator anyway so for all of us it just looks foreign. The Chrono library makes some weird choices in the API. There was no need to brush the API that much imho

permalink embed save parent report reply

[–] **RamielIsMyWaifu** 5 points 1 month ago

> What's the motivation for this?

it looks cool

permalink embed save parent report reply

load more comments (6 replies)

[–] **manugildev** 106 points 1 month ago

Break the ABI and save C++

permalink embed save report reply

[–] **_VZ_** wx | soci | swig 9 points 1 month ago*

Why are so many people upset about not breaking ABI? Is the existence of a (de facto) ABI really such a big problem? If so, how/why exactly?

Edit: It seems my question was misunderstood, so let me try to clarify. I understand the advantages of keeping the ABI and the problems inherent to breaking it. What I don't understand is why are there several comments just in this thread complaining about *not* forcefully breaking it. IMO this is really not the most urgent problem to solve in C++, while the problem with `unique_ptr` is indeed annoying, I just can't imagine there are that many people who absolutely can't live with it or apply some workaround. So my question was why do people asking for breaking the ABI do it and what exactly do they hope to gain by this.

permalink embed save parent report reply

[–] **shotashotshotashota** 14 points 1 month ago

There's this law that says every observable behavior of a system will be used by *someone*. Even if its a bug, if it existed long enough, they becomes a feature that someone, somewhere, uses.

permalink embed save parent report reply

[–] **daveedvdv** EDG front end dev, WG21 DG 12 points 1 month ago

Lately, that's been referred to as Hyrum's Law (after an engineer at Google, I believe).

permalink embed save parent report reply

[–] **kkert** 8 points 1 month ago

This should answer it, more or less: wg21.link/P2028 (shorter: wg21.link/P1863)

permalink embed save parent report reply

load more comments (13 replies)

[–] **James20k** P2005R0 29 points 1 month ago

So, in a lot of fields, yes. Its not uncommon for a library vendor to provide a C++ library which is closed source (eg the steam API dll), which your application is built against. If there is an all or nothing ABI break, in C++23 mode your application will now fail to compile against that closed source DLL

In a large organisation, it is a massive amount of work to fix ABI issues, because a large change like this would have to be coordinated between vendors and people using code. When you have 10s of millions of lines of code, this is fairly impractical, unless you're google

Then there are the closed source binaries for which we no longer have source, which means that an ABI break is super problematic

Its worth noting that C++ *does* break ABI - but compiler vendors have tricks that they can use to mitigate the impact to users, aka everything is fine. One of the big things that came out of the discussion around ABI is that vendors have a lot more power here than people traditionally think they do to mitigate smaller ABI breaks, so we should consider proposals that do contain ABI breaks instead of dismissing them as has been done traditionally

I'm not convinced there isn't a toolchain solution here, where people who want ABI stability can have eg clang generate a shim, which is one of my projects to look at post prague

The main argument *for* breakage is performance, particularly unique_ptr by value and std::unordered_map

permalink embed save parent report reply

[–] **mpyne** 19 points 1 month ago*

To add to this, even open-source projects written using C++ often have policies on maintaining ABI. This is the case with KDE.org for instance.

There's a lot of reasons for this but the primary one is that our users aren't able to recompile their entire Linux distro every hour on the off chance that there was a source-compatible ABI break in a base library that we provide. Ensuring ABI is maintained across releases within the same major version is *what makes it possible at all* for our users to safely upgrade to new patch or minor releases without breaking all of their other software. This allows for smaller changes and more effective testing of those changes.

This is also one of the reasons that we sometimes use Qt versions of types or libraries that seem to have viable 'native C++' equivalents. I *know* that `QString` in Qt5 is going to be forward-compatible at an ABI level for the entire supported timeframe of Qt5, but I can't say the same for `std::string`.

In a way this is almost an argument to choose to break ABI with a given C++ release (along the lines of the upcoming Qt5 -> Qt6 transition where we know ABI will break), but that type of planned ABI break is only useful if there's a semblance of a guarantee of ABI stability afterwards until a subsequent announced break, and I haven't seen anyone pushing for that.

An ABI break is incredibly disruptive, the more so as the ABI becomes lower and lower level and should not be considered lightly. KDE has a KDE Frameworks 5 library called kdelibs4support which does more or less what it says on the tin, and we still have applications which use that ABI upgrade path support library nearly a decade later. I don't even want to think about how working but unmaintained software would handle an ABI break at the base C++ language/runtime layer!

permalink embed save parent report reply

[–] **FrankHB1989** 4 points 1 month ago*

It is reasonable to provide compatibility over a limited set of binary configurations of the systems, mainly for *end-users*. It is reasonable to require a library reusable without rebuilding them, which also (hopefully) saves other developers' work.

However, technically, such features are not implementable without restrictions, since there are too many things (e.g. any machine-specific compilation options) out of the control from the project maintainers. Only the publisher of the binary libraries (distributions) can eventually verify and ensure such compatibility features for users.

Restrictions on the source code can make the resulted binaries more predictable, hence it needs less work of the library publishers. For the binary compatibility defined here, this is a *workaround* rather than a *solution*. Although such best-effort is often a good practice, it is not a must for all cases, and sometimes even harmful.

permalink embed save parent report reply

load more comments (1 reply)

[–] **FrankHB1989** 10 points 1 month ago*

> The main argument _for) breakage is performance, particularly unique_ptr by value and std::unordered_map

Not quite true. There is actually nothing to prevent objects of `std::unique_ptr<T>` passed by register technically in current C++. The fact is, *specific ABIs* used by some popular implementations prevent it to be done. (Note that it does not prevent aggressive optimizations across TUs.)

So, the real argument here is to ease the work of implementations at the cost of users, although users of the language will gain some expressiveness from this specific resolution. However, it is still a shame to blindly attribute those QoI issues to the so-called ABI breakage problem and to expect them resolved totally in the high-level language design.

For users of the language, there is one true need for the breakage: to make it fail fast and to get rid of the bug-to-bug compatibility endorsed by the false guarantees in a more explicit way. There are merely a few comparability features provided by documented ABI specs (e.g. involving ISA-specific interoperations). Relying on things beyond those features are totally nonsense for average C++ users who have no effort to dig deep into the implementations once they meet weird problems. They are away from sane and predictable interactions of the implementations, almost as bad as relying on undefined behaviors. (Those relying on blobs deliberately are deserved to get the risks of the breakage anyway; that is another story.)

permalink embed save parent report reply

[–] **SkoomaDentist** Antimodern C++, Embedded, Audio 9 points 1 month ago

> Its not uncommon for a library vendor to provide a C++ library which is closed source (eg the steam API dll), which your application is built against. If there is an all or nothing ABI break, in C++23 mode your application will now fail to compile against that closed source DLL

This is misleading, particularly as you use DLLs as example. There never has been a stable C++ ABI on Windows. At most the ABI is no longer broken between every major compiler release, but there is no expectation about long term C++ ABI stability on Windows. That's just the nature of the beast.

The platform where there has been built up such expectation is the one where it should be needed the least, namely Linux (and other *nixes), as the source code is almost always provided. In fact the entire problem is largely selfmade since the stdlib maintainers have had the habit of not breaking the ABI (with a major exception being std::string).

So now the entire C++ language is held hostage due to the implicit expectations of a single platform and for some unfathomable reason people are defending this state of things. A bizarre situation indeed.

Also we should remind people that there is no such thing as "the C++ ABI". Or can someone point me to the part in the C++ standard that defines such thing?

permalink embed save parent report reply

[–] **Plorkyeran** 3 points 1 month ago

The Microsoft C runtime historically did not provide a stable ABI, but the C++ ABI on Windows has been stable forever and providing a DLL which works with every version of vc++ is not very hard. You mostly just can't expose any standard library types in your API and have to ensure that everything allocated by your DLL is also deallocated by your DLL.

permalink embed save parent report reply

[–] **SkoomaDentist** Antimodern C++, Embedded, Audio 4 points 1 month ago

As far as stdlib ABI breakage (which is what's really being discussed here) goes, the result is still the same: Microsoft can (and will) break the ABI when it deems necessary and people aren't going to complain much as long as it doesn't happen between every major compiler version.

permalink embed save parent report reply

[–] **arclovestoeat** 2 points 1 month ago

How common is it to ship binary-only C++ libraries? In binary form, I've mostly only dealt with C libraries, or very pared down C++ (eg, no std::string in interface). Could things still break if the binary library linked against an old standard library?

permalink embed save parent report reply

[–] **SkoomaDentist** Antimodern C++, Embedded, Audio 6 points 1 month ago

DLLs can use different versions of stdlib on Windows. So as long as your public API is C (with possible a client side compiled C++ wrapper to make it nicer to use), ABI breakage is a nonissue.

permalink embed save parent report reply

[–] **manugildev** 12 points 1 month ago

IMO, people want to stick to C++ but they don't want the new features, they seem patches.

C++ has made lots of mistakes during the years, bad design choices that can not be break because that would require a change of the ABI, and therefore breaking the language.

Is hard that a 40yo language evolves into a modern one, even more if so many legacy systems and programmers rely on it.

permalink embed save parent report reply

load more comments (16 replies)

[–] **meneldal2** 4 points 29 days ago

> while the problem with unique_ptr is indeed annoying

Isn't that entirely the implementation fault? As is, the standard itself doesn't make it inefficient, the implementation just doesn't deal with it well.

permalink embed save parent report reply

load more comments (8 replies)

[–] **foonathan** 9 points 1 month ago

A big concern are companies that only sell compiled C++ code and have since then gone out of business, so nobody has the source code anymore to recompile. If there is an ABI break, people using such products are basically screwed.

permalink embed save parent report reply

[–] **kkert** 23 points 1 month ago

> If there is an ABI break, people using such products are basically screwed.

No, they are not. It is always possible to wrap the functionality with your old compiler into a more stable interface. Either put C ABI around your component, wrap it in separate executable altogether, expose it over some interface like DBUS or COM, or just make it a web service.

permalink embed save parent report reply

load more comments (3 replies)

[–] **mcencora** 34 points 1 month ago

They are screwed regardless of ABI break - recent types of security vulnerabilities like spectre/meltdown are best proof.

permalink embed save parent report reply

[–] **manugildev** 17 points 1 month ago

They only have to use old compilers, that's it.

permalink embed save parent report reply

load more comments (1 reply)

[–] **BenFrantzDale** 5 points 1 month ago

Would it be possible to have a translation layer between ABIs? At least if the library boundary weren't performance-critical?

permalink embed save parent report reply

[–] **mjcaisse** 3 points 1 month ago

No. Said companies just don't update compilers. Industry isn't usually as eager to move compilers in a shipping product as many people would have you think.

permalink embed save parent report reply

load more comments (1 reply)

[–] **kalmoc** 3 points 1 month ago

I think the whole std::unique_ptr problem is highly overrated. However, there are easily a dozen small and big things that could be improved through the standard library (both in terms of specification and implementation) that are blocked on ABI stability.

Now, I'd prefer to have to deal with a single ABI break point across the eco system every 9-12 years than multiple ABI breaks over time or total stagnation.

permalink embed save parent report reply

load more comments (35 replies)

[–] **tvaneerd** C++ Committee, lockfree, PostModernCpp 20 points 1 month ago

See Bryce, I told you C++ was done. (well 20 at least)

permalink embed save report reply

[–] **sempuki** 9 points 1 month ago

Post Modernist. Creative director.

permalink embed save parent report reply

load more comments (1 reply)

[–] **funnypigrun** 29 points 1 month ago

C++20 was just finished but I'm already excited for C++23! The future of C++ looks promising.

permalink embed save report reply

[–] **GerwazyMiod** 4 points 1 month ago

Right? Executors and Networking!

permalink embed save parent report reply

load more comments (1 reply)

[–] **[deleted]** 13 points 1 month ago

I might be stupid, so can anyone explain to me the difference between "move = bitcopies" by Niall Douglas and "Object relocation in terms of move plus destroy" by Orthur O'Dwyer? From what I can tell, both are talking about "destructive move operations". Also what's the reason for this poll? I get that the papers have different approaches, but aren't they solving the same problem?

permalink embed save report reply

[–] **Dragdu** 8 points 1 month ago

One aims at providing library facilities that user's can opt-in and implementations can leverage, while the other wants to modify how a fundamental operation works in the language.

permalink embed save parent report reply

[–] **[deleted]** 2 points 1 month ago

Okay, that makes sense. Still, why would they progress independently?

permalink embed save parent report reply

[–] **14ned** LLFIO & Outcome author | Committees WG21 & WG14 3 points 1 month ago

P1144 enables standard library containers to be less stupid with collections of some types. It does not modify ABI of such types otherwise e.g. return of them from functions.

P1029 is the opposite almost: types opted into move bitcopying get improved codegen i.e. ABI break over if they were not opted in.

Both proposals enable standard library containers to be less inefficient, however P1144 produces superior efficiency improvements to P1029 for standard library containers.

As both proposals are orthogonal (one never breaks ABI, the other explicitly is for breaking ABI), EWG-I has voted twice now to recommend they be progresed separately. P1144 has gone to EWG, P1029 should go to EWG next meeting I would expect.

permalink embed save parent report reply

[–] **[deleted]** 2 points 1 month ago

Thanks for the explanation. That clears it up.

permalink embed save parent report reply

[–] **nemanjaboric** 3 points 1 month ago

The way I see this is since we don't have any evidence if one of them would make the way into the language and since they are sufficiently different (they take completely different approaches) it doesn't make much sense to try to shoehorn them together. Similarly, waiting to see if one would fail and then pursuing the other may be a waste of time.

permalink embed save parent report reply

load more comments (1 reply)

[–] **hachanuy** 46 points 1 month ago

> Many in the committee are interested in considering targeted ABI breaks when that would signify significant performance gains.

That raised hope a bit then this struck

> Notably, the proposed epochs language facility received no consensus to proceed.

Nooo...

permalink embed save report reply

[–] **famastefano** 23 points 1 month ago

Well they said that epochs have some issues 🤷‍♂️

It's a big proposal after all, it's hard to think about every single possible problem.

permalink embed save parent report reply

[–] **hachanuy** 15 points 1 month ago

I know the committee wouldn't reject it for no reason but it still stings that there's a major problem with epoch. I hope that can be fixed when more understanding about module and concept is gained.

permalink embed save parent report reply

[–] **chuk155** graphics engineer 14 points 1 month ago

A unready proposal accepted into the standard is forever bad. There being large scale issues with the design is in inevitable, its trying to do a very large thing. In fact, if it did somehow sail through something is deeply wrong with the committee. And there are still 1-2 years worth of meetings for it to make it into C++23, so don't give up hope just because it isn't perfect from the get go.

permalink embed save parent report reply

[–] **HappyFruitTree** 2 points 1 month ago

Does *"no consensus to proceed"* mean they will look at it again?

permalink embed save parent report reply

[–] **chuk155** graphics engineer 3 points 1 month ago

It means "We didn't reject it but don't think its ready for the next committee in the process" (Language Evolution in this case).

So yes, come next meeting they will hopefully look at it again, especially if it has received changes in that time.

permalink embed save parent report reply

[–] **D_0b** 11 points 1 month ago

> If one TU thinks C<T>is true, but another TU in a later epoch thinks C<T> is false, that easily leads to ODR violations.

Can anyone give a concrete example how this can happen?

permalink embed save report reply

[–] **D_0b** 9 points 1 month ago

A situation where the break is happening is probably with the first example in the paper removing implicit conversions for builtin types.

if you have a concept that tries to call a function that would use implicit conversion in an old TU it will be true, in a new TU it will be false.

permalink embed save parent report reply

[–] **SuperV1234** 2 points 1 month ago

(paper author here) This is a possible outcome, but we could also decide to make epoch restrictions behave as a glorified -Werror switch. As an example, we could stop compilation immediately when an implicit conversion is detected rather than change the outcome of SFINAE or overload resolution.

permalink embed save parent report reply

load more comments (6 replies)

[–] **bigcheesegs** Tooling Subgroup (SG15) Chair | Clang dev 9 points 1 month ago

The example I gave during the discussion was `std::is_constructable_v`. One of the examples in the paper was removing implicit conversion. The problem with this is that you have three options for the behavior of templates, and all of them are bad.

```
epoch 2023; // Module-level switch
export module Particle;
import <type_traits>;

export struct Particle {
  Particle(float x, float y);
  float x, y;
};

export void example() {
  if constexpr (std::is_constructible_v<Particle, double, double>)
    Particle(1.2, 4.8);
}
```

1. Use the epoch of the instantiation context: ODR violation when there's another instantiation in a previous epoch.
2. Use the epoch of the template definition: The trait lies, and you get an error even though you checked first.
3. Use the epoch of the owning module of `Particle`: The behavior of the language doesn't depend on the epoch of the current module. Also more complicated with mixing multiple types from different epochs.

This isn't a problem for every possible change you could introduce in an epoch, but it is for everything interesting I've seen discussed.

permalink embed save parent report reply

[–] **Lyberta** 3 points 1 month ago

I think this can be fixed by breaking the assumption of TUs being independent or require all potentially ODR violating code being marked and put into separate section that linker must check for duplicates and error out if dupes exist.

permalink embed save parent report reply

[–] **bigcheesegs** Tooling Subgroup (SG15) Chair | Clang dev 2 points 1 month ago

> I think this can be fixed by breaking the assumption of TUs being independent

Not sure what you mean by this.

> require all potentially ODR violating code being marked and put into separate section that linker must check for duplicates and error out if dupes exist.

Almost all code is potentially ODR violating, and that just makes your code super fragile. Spooky action at a distance is bad.

permalink embed save parent report reply

[–] **MartenBE** 3 points 1 month ago

Is there a movement to address the issue epochs tried to resolve, or will it be ignored for the time being?

permalink embed save parent report reply

[–] **SuperV1234** 2 points 1 month ago

I think you're missing one options we briefly mentioned in EWGI: https://www.reddit.com/r/cpp/comments/f47x4o/_/fhslry8

It has it's own weirdness though, because the trait would evaluate to true but you wouldn't be able to invoke a constructor, which is inconsistent...

permalink embed save parent report reply

[–] **therealcorristo** 6 points 1 month ago*

Say you have a header file that defines the following struct `struct foo { void bar() { std::puts("bar called"); } };`

and a concept that requires that a function `bar` can be called on a `const` object. Then any TU compiled with current defaults will see `foo::bar` as a non-const member function, i.e. the concept is not satisfied, while any TU using a new epoch that makes member-functions `const` by default will see `foo::bar` as a `const` member function, i.e. the concept is satisfied in that TU.

permalink embed save parent report reply

[–] matthieum 16 points 1 month ago

This would indeed be terrible, however I would argue that this is not how epochs should work.

Instead, I would argue that the rules that apply to an item should be the rules that apply to the module the item is defined in. That is:

- If `foo` is defined in a C++20 module, then it follows C++20 rules even when used in a C++23 module.
- If `foo` is defined in a C++23 module, then it follows C++23 rules even when used in a C++20 module.

Thus in this case, if C++23 "infer" constness and C++20 doesn't, this does not lead to issues -- no matter where it is used, a single item obeys a single set of rules.

permalink embed save parent report reply

[–] therealcorristo 2 points 1 month ago

You're probably right, I missed that `foo` is either reexported in both modules that include the header, in which case the ODR violation already happens at that point if the different epochs cause the definition of `bar` to differ, or `foo` is included in an implementation file of at least one of the modules in which case it has module linkage and the two `foo`s aren't the same struct.

permalink embed save parent report reply

[–] matthieum 7 points 1 month ago

On the other hand, such a scheme does require a compiler which implements *all rules*. It's already the case today -- with compilers having switches to choose the standard version -- so doesn't seem a problem, but it does mean compilers are bound to only get bigger and bigger over time.

Also, there are also questions as to what rules should template use:

```
template <typename T>
void quadsort(T* begin, T* end);
```

If `quadsort` is defined in C++20 module but is instantiated with a `T` from a C++23 module, which set of rules applies? Can this lead to issues?

Note: in Rust, epochs are only used for syntactic constructs -- mainly introduced new keywords -- which is super easy. Having epochs impact semantics or ABI is very much untested ground.

permalink embed save parent report reply

[–] **[deleted]** 8 points 1 month ago

> Having epochs impact semantics or ABI is very much untested ground.

Epochs (the proposed C++ ones, not rust ones) wouldn't touch ABI, but I believe "we can mess with semantics" was advertised as one of the strong points. It's definitely hairy.

permalink embed save parent report reply

load more comments (1 reply)

load more comments (2 replies)

[–] D_0b 7 points 1 month ago

from what I understood of epochs, if a struct is defined in the new epochs, they are not suppose to "see" them as const but actually be marked const. (so both old and new will see them as const) on the other hand if foo is defined in an old epoch it is as it is, and both old and new will see it as non const.

permalink embed save parent report reply

load more comments (8 replies)

[–] **jcelerier** ossia score 32 points 1 month ago

> However, concerns were raised about security and usability problems, so the ability to execute arbitrary code at compile-time was rejected.

I wonder why it's a problem for C++ and not for so many other languages - every interpreted one for starters, but also things like F#, Zig...

permalink embed save report reply

[–] **SeanMiddleditch** Game Developer 4 points 28 days ago

Languages like Zig have a fraction of the userbase and possibly zero devs who actually *care* about build environment security and such.

C++ is used by orders of magnitude more people and in more sensitive environments, and there are folks in the committee who deeply care about things like whether a third-party library could hijack an internal build node or whether an internal dev could use it to copy CI machine tokens/passwords or so on.

(I don't think F# has compile-time code arbitrary code execution with I/O... does it?)

permalink embed save parent report reply

[–] **foonathan** 8 points 1 month ago

Another huge concern is related to cross compiling. Currently, the constexpr interpreter emulates the target platform completely. The circle model executes native code. This means that when cross compiling, sizeof() in compile time and runtime code might have different values, floating point evaluation differs, etc. etc.

permalink embed save parent report reply

[–] **seanbaxter** 32 points 1 month ago

Not true. Circle adopts the architecture and abi of the target. sizeof reflects the target. The only cross compilation complication is executing foreign function calls on the host. The committee should have decided to not include foreign function calls, and they would have gotten everything else, like full C/C++ library access at compile time.

permalink embed save parent report reply

load more comments (8 replies)

load more comments (2 replies)

[–] **tcanens** cppreference.com | LWG 9 points 1 month ago*

A slightly fuller summary of what we did in LWG, in addition to what was mentioned above:

- Renaming galore:
  - `safe_range` was renamed to `borrowed_range` (likewise for `safe_iterator_t` etc.)
  - `default_constructible` was renamed to `default_initializable`
  - `all_view` was not really a view type and was renamed to `views::all_t`.
  - `leap` was renamed to `leap_second`, and `link` was renamed to `time_zone_link`
  - `*_default_init` was renamed to `*_for_overwrite`
  - `ispow2`, `ceil2`, `floor2` and `log2p1` were renamed to `has_single_bit`, `bit_ceil`, `bit_floor` and `bit_width`, respectively.
  - Range algorithm result types were renamed with the old names becoming aliases, e.g., `copy_result` became `in_out_result`; `partition_copy_result` became `in_out_out_result`.

- `span` got more things ripped out and other things adjusted
  - `cbegin` and friends are removed
  - tuple-like protocol (including structured binding support) for fixed-size spans is removed
  - fixed-size span's constructor from dynamically-sized ranges is now explicit (size mismatch is still undefined)
  - construction from `std::array` now allows qualification conversions
- `std::boolean` was removed and replaced with an exposition-only `boolean-testable` concept.
- We added `ranges::` versions of `for_each_n`, `clamp` and `sample`. We found an issue with the proposed `ranges::shift_left` and `ranges::shift_right` so they had to be kept back.
- Lots of bug fixes. We applied 109 (!) issue resolutions directly. A list can be found here - everything in "Voting" or "Immediate" status was applied. A number of the adopted papers also fall into this category. A few notable ones not already mentioned:
  - `has_strong_structural_equality` is removed now that "strong structural equality" is no longer a thing.
  - `std::pair` and `std::array` are guaranteed to be usable as the type of non-type template parameters (if the element type(s) are themselves usable as such)
  - rvalue stream operations now preserve the type of the stream, so that you can write `(std::ostringstream() << "i = " << i).str()`

permalink embed save report reply

[–] **kalmoc** 3 points 29 days ago

> ispow2, ceil2, floor2 and log2p1 were renamed to has_single_bit, bit_ceil, bit_floor and bit_width, respectively.

Why?

> tuple-like protocol (including structured binding support) for fixed-size spans is removed

Why?

permalink embed save parent report reply

[–] **tcanens** cppreference.com | LWG 11 points 29 days ago

`log2p1` collides with an IEEE754 function that has completely different semantics. More generally, LEWG wanted to later extend these functions to things like `std::byte` which have no mathematical operations, so the previous names are not ideal.

There's a late-breaking design issue with the definition of `tuple_element_t<0, span<int, 42>>`, and LEWG decided to remove it rather than trying to fix it in <10 hours.

permalink embed save parent report reply

load more comments (1 reply)

[–] **barchar** MSVC STL Dev 4 points 28 days ago

because we love changing spelling. It's fun!

permalink embed save parent report reply

load more comments (1 reply)

[–] **JoelFilho** 8 points 1 month ago

Congratulations to all involved!

I definitely can't wait to write my libraries in modules without ugly SFINAE template code and cryptic compiler errors.

Also, since I'm unfamiliar with how the committee works, a (hopefully inoffensive) question:

> SG12 also collaborated with the MISRA standard for coding standards in embedded systems to help them update the guidelines for newer C++ revisions.
>
> The freestanding library took a few steps forward, with some interesting proposals, including Freestanding Language: Optional ::operator new
>
> One of the biggest decisions was on Low-Cost Deterministic C++ Exceptions for Embedded Systems which got great reactions. We will probably hear more about it!

Since embedded systems have some importance as an unique field for the language, how much more would the committee need to create a dedicated embedded/freestanding study group?

permalink embed save report reply

[–] **ben_craig** freestanding 8 points 1 month ago

Arguably, SG14 (Low Latency) covers a lot of this territory. I tend to present my freestanding papers there first before going to the respective incubators.

permalink embed save parent report reply

[–] **JoelFilho** 3 points 1 month ago

Thanks for answering. I thought Low Latency focused more on fields like high-frequency trading, but it makes sense that it could also do embedded without requiring a new SG.

Also, thanks for your work on the freestanding proposal. Coming from embedded, I can't count the amount of "C++ shouldn't be used on embedded"-like thinks I've heard because of the usual complaints about heap and exception usage in the STL. Hopefully the implementation of these proposals will increase the adoption of modern C++ in the future of embedded software development.

permalink embed save parent report reply

load more comments (2 replies)

[–] **Adverpol** 9 points 1 month ago

Really excited to see the proposed changes. Really bummed that it might take 6 years before we get pattern matching. 6 years just feels crazy long, I mean it's a substantial part of the total time-frame of my career as a C++ dev.

permalink embed save report reply

[–] **c0r3ntin** 4 points 29 days ago

My money is on it shipping in 23

permalink embed save parent report reply

[–] **Adverpol** 3 points 28 days ago

My hopes and dreams are ;)

permalink embed save parent report reply

[–] **seanbaxter** 18 points 1 month ago

I implemented pattern matching in my compiler. Took all of two weeks.

https://github.com/seanbaxter/circle/blob/master/pattern/pattern.md

If they're "looking for implementation experience" they aren't doing a good job, since I've had it ready since September and nobody from wg21 has asked me one thing about it. If you want to see advanced features, the future of C++ does not belong to ISO.

permalink embed save parent report reply

[–] **sempuki** 13 points 1 month ago

Clang could implement a non standard, non portable, unstable version in a couple weeks too. Let us know when you get your first thousand full time developers -- or your first billion dollar user. Snarky snipes from the sidelines aren't helpful. Software is hard.

permalink embed save parent report reply

[–] frog_pow 8 points 1 month ago

Circle looks very impressive, I wish it had been given more serious consideration, hopefully it will be given a more fair look again in the future..

permalink embed save parent report reply

[–] Adverpol 3 points 29 days ago

Wow, that looks awesome. It also makes 6 years that much harder to swallow. Even if, like u/sempuki says, this is not battle-tested, a usable implementation seems to me to be the first step on that road. If this would be like rust-nightly then we could actually have people use it and get real-world feedback, even if if's only for hobby projects.

permalink embed save parent report reply

load more comments (1 reply)

[–] kalmoc 31 points 1 month ago

> Given the current state of std::regex is such that we cannot fix either its interface or its well-known performance issues, a number of volunteers agreed to bring a paper to deprecate std::regex at a future meeting.

So deprecation and eventual removal are preferable to fixing, but breaking ABI? That has to be a joke right?

permalink embed save report reply

[–] sempuki 8 points 1 month ago

Deprecation will last a long time.

permalink embed save parent report reply

[–] Dascandy HippoMocks/cpp-dependencies/Evoke/Pixel dev 5 points 1 month ago

Explicitly not removal.

permalink embed save parent report reply

[–] mort96 11 points 1 month ago

The eventual goal of a depreciation surely is removal, right? Like how auto_ptr has been deprecated forever and finally got removed in 17

permalink embed save parent report reply

[–] Dascandy HippoMocks/cpp-dependencies/Evoke/Pixel dev 13 points 1 month ago

The goal with this deprecation is to make clear to everybody that:

- It has issues. Very big issues. Big enough that you should not want to use this in any new code, and consider removing it in existing code.
- We know it has issues. It's not news, at least for most of the issues.
- We know that we cannot get any fix through the whole committee. We've tried a few times and in some different ways.

Given that this is not a good solution and we know you should never want to use it, it should be deprecated.

It *cannot* be removed, until we have a replacement. It will only be removed after the replacement has had time to replace people's use of std::regex.

Right now though, we really want people to understand that you shouldn't use this, and we're not helped (in fact - the replacement would take longer!) if people keep adding suggestions to fix it.

Think auto_ptr in a hypothetical C++08. Yes, it'd get a replacement in 11, and it would be removed in 17, but we'd want to tell you "don't use this, and don't submit papers with fixes" in 08.

This paper's goals are the deprecation. Removal is a different paper, the replacements are different papers.

permalink embed save parent report reply

[–] **PeterBrett** SG16; CAD software dev 5 points 1 month ago

Is it okay if I cut & paste this into my draft paper?

permalink embed save parent report reply

[–] **Dascandy** HippoMocks/cpp-dependencies/Evoke/Pixel dev 3 points 1 month ago

Already have a draft. Will send it to you and Hana in a few hours.

permalink embed save parent report reply

[–] **kalmoc** 3 points 1 month ago

Why do we need a regex library in the standard at all?

permalink embed save parent report reply

[–] **PeterBrett** SG16; CAD software dev 10 points 1 month ago

> Why do we need a regex library in the standard at all?

Because I have spent ~50% of my software engineering career fixing obscure buffer overruns in hand-written lexers which didn't use regular expressions because the author hated adding dependencies.

permalink embed save parent report reply

load more comments (3 replies)

[–] **c0r3ntin** 3 points 1 month ago

Except very rare cases, depreciations are not intended to lead to removal

permalink embed save parent report reply

[–] **STL** MSVC STL Dev 12 points 1 month ago

That was a fairly accurate description of the status quo circa C++98-11, but nowadays, deprecated features are regularly checked for being candidates for removal, and many C++17-deprecated features were duly removed in C++20. Which is good.

permalink embed save parent report reply

[–] **c0r3ntin** 5 points 1 month ago

We talked about this this week and I think there is some consensus (no poll) that nothing should ever be removed unless actively harmful (like auto_ptr).

I still don't know how I feel about that.

A few months ago, a proposal I had to mark the thing in annex D with deprecated failed spectacularly

permalink embed save parent report reply

[–] **GerwazyMiod** 5 points 1 month ago

Kill it with fire, start fresh with CTRE.

permalink embed save parent report reply

load more comments (2 replies)

[–] **James20k** P2005R0 7 points 1 month ago

Nope, std::regex is apparently completely unfixable without major ABI problems

permalink embed save parent report reply

[–] **ROYAL_CHAIR_FORCE** 6 points 1 month ago

Sorry might be a stupid question, but what are the problems with the regex API?

permalink embed save parent report reply

[–] **Dascandy** HippoMocks/cpp-dependencies/Evoke/Pixel dev 19 points 1 month ago

Searching for only á will get you í too. Maybe. Sometimes ý too. Or hit á three times. And find a too. But not always. Could find ç. Or some chinese characters and emoji.

The actual performance is... pretty bad.

It supports 7 language variants.

None of this can be touched on at least one compiler without a total ABI break.

permalink embed save parent report reply

[–] **STL** MSVC STL Dev 10 points 1 month ago

> It supports 7 language variants.

That's an egregious exaggeration! How dare you besmirch the good name of basic_regex? There are only *6* grammars: ECMAScript, basic, extended, awk, grep, egrep.

(This is a joke - there are indeed way too many grammars and only ECMAScript should exist.)

permalink embed save parent report reply

[–] **[deleted]** 7 points 1 month ago

The ECMA grammar, at least as adopted by `std::regex`, doesn't support multiline patterns as specified in `std::regex::multiline`. This made me resort to patterns like `(?:\r|\r\n|\n|$)` instead of just `$` which worked in `boost::regex`.

For the record, I agree that having 97 grammars is way too many. I'm just playing the devil's advocate.

permalink embed save parent report reply

load more comments (4 replies)

load more comments (1 reply)

[–] **Lyberta** 7 points 1 month ago

That's because you're trying to use `std::regex` with Unicode which was never supported in the first place.

permalink embed save parent report reply

load more comments (1 reply)

load more comments (1 reply)

[–] **HappyFruitTree** 3 points 1 month ago

Removal doesn't necessarily break code because implementations could continue to support it indefinitely. A breaking change would force code to break.

permalink embed save parent report reply

load more comments (1 reply)

load more comments (1 reply)

[–] **Sartrean010** 7 points 1 month ago

Can members of the public show up just to observe if you don’t represent a company or organization?

permalink embed save report reply

[–] **14ned** LLFIO & Outcome author | Committees WG21 & WG14 12 points 1 month ago

You need to register beforehand so they can allocate space for you, but otherwise, yes.

permalink embed save parent report reply

[–] **Sartrean010** 3 points 1 month ago

Cool. :D

permalink embed save parent report reply

[–] lctogan 6 points 1 month ago

Did pr1105 make any progress? I'd really like if things that are common practice for embedded development like the lack of exceptions or rtti would get adopted by the standard.

permalink embed save report reply

[–] foonathan 14 points 1 month ago

Sort of, the freestanding proposals made progress, so more and more stuff of the standard library can be available on certain embedded platforms. This subset does neither exceptions, nor heap allocations or RTTI.

LEWG also voted to make a lot more library functions noexcept (which isn't a semantic change; those functions weren't throwing before, they just weren't noexcept for ... reasons).

permalink embed save parent report reply

[–] ben_craig freestanding 8 points 1 month ago

Bad news: I have no plans on making further revisions to the P1105 omnibus paper.

Good news: The respective pieces of P1105 will be getting papers. In particular, P2013 (Optional ::operator new) has been received favorably (no against votes at all!). I was instructed to write some wording and bring it back to EWG.

On the library front, both P1641 and P1642 were received well in Library Incubator. These are some of the "little pieces" of P0829.

permalink embed save parent report reply

[–] tambry 6 points 1 month ago

> treat initialization of a bool from a pointer as narrowing

Links to the same paper as concept value caching.

But this is good. A co-worker recently wrote a bug that would have been prevented by this.

permalink embed save report reply

load more comments (1 reply)

[–] adamgetchell 26 points 1 month ago

Really disappointed with the decision re: Circle. The rationale, if I understand correctly, “the compiler might execute untrusted code”, seems to be an orthogonal problem that already exists with current compilers. [1] [2]

Seems like a heavy burden to impose on a meta programming framework.

Seems if that was truly an overriding concern there would be more work done on integrating formal verification. [3]

Meanwhile, we as the community are losing out on true innovation that will make our programming immediately better while keeping (and even improving) the “bare metal performance “ of C++.

[1] https://www.schneier.com/blog/archives/2006/01/countering_trus.html [2] https://www.archive.ece.cmu.edu/~ganger/712.fall02/papers/p761-thompson.pdf [3] http://compcert.inria.fr

permalink embed save report reply

[–] SAHChandler C++ Bruja 20 points 1 month ago

I like also how that is the excuse given, when

1. No discussion of a threat model has been given
2. No one is verifying their build systems aren't executing untrusted code
3. No one is verifying their compiler wasn't built on a compromised machine

This "what about security concerns?" approach to arguments seems to always end in hand waving, but no one ever discusses to what degree their threat model concern is (probably because with enough prodding and poking you'd be able to point out that their perceived threat model is already a problem with the status quo)

permalink embed save parent report reply

[–] **Janos95** 8 points 1 month ago

Having Code that can do foreign function calls at compile time is more dangerous if and only if one never intends to run the code. Only compiling without running code seems like a very niche market to me ;)

permalink embed save parent report reply

load more comments (1 reply)

[–] **sempuki** 2 points 1 month ago

The committee doesn't have the capacity to do proper security analysis. Hopefully that can change in the near future.

Sign up your local security expert to attend.

permalink embed save parent report reply

[–] **SAHChandler** C++ Bruja 5 points 29 days ago

If they don't have the capacity to do security analysis then maybe using the phrase "there are security concerns" as a metaphorical boogie man should be dismissed.

permalink embed save parent report reply

[–] **sempuki** 2 points 29 days ago

Which do you prefer with regard to security, false positives, or false negatives?

permalink embed save parent report reply

[–] **SAHChandler** C++ Bruja 4 points 28 days ago

I prefer that people not use the phrase "security" as a boogie man argument and try to express what their actual concerns are instead of trying to subjectively kill a paper because they're afraid of it.

permalink embed save parent report reply

[–] **sempuki** 3 points 28 days ago

Good thing that's not what happened then.

permalink embed save parent report reply

[–] **c0r3ntin** 6 points 1 month ago

Beside security, it would be such a can of worm that we would not make progress on reflection in the next decade. Circle model is basically 2 coexisting abstract machines.

permalink embed save parent report reply

[–] **TheSuperWig** 6 points 1 month ago*

Partially mutable lambda captures is listed twice. Improving Engine Seeding and Portable Distributions link to the same paper.

permalink embed save report reply

[–] **sempuki** 4 points 1 month ago

Thank you, that is my mistake, but I think only u/blelbach can fix it.

permalink embed save parent report reply

[–] **encyclopedist** 5 points 1 month ago

While we are at it, it seems that sentenses about "freestanding" and "deterministic exceptions" are in the wrong section. Currently these are in the "Machine learning" section.

permalink embed save parent report reply

load more comments (1 reply)

[–] **TheSuperWig** 6 points 1 month ago

Also "a new `status_code` facility"'s link has a typo. It links to p1208 instead of p1028.

cc /u/blelbach

permalink embed save parent report reply

load more comments (1 reply)

load more comments (1 reply)

[–] **rlamarr** 7 points 1 month ago

Hurray!

permalink embed save report reply

[–] **johannes1971** 5 points 1 month ago

Question: will the "inline in modules" rule affect actual inlining? I.e. will we revert from the current situation where the compiler decides what to inline, to us having to specify this ourselves again?

permalink embed save report reply

[–] **cpp_learner** 3 points 1 month ago*

AFAIK compilers cannot inline a function that is defined in a different TU (translation unit), and a module is a different TU, so it's not very different from the current situation.

A linker can of course choose to inline the function call at link time, though.

permalink embed save parent report reply

[–] **johannes1971** 2 points 1 month ago

My concern is functions that you currently define (physically) inline, like all those rather minimal setters and getters that (I assume) are just going to be optimized away completely currently. Will we have to mark those with the actual keyword `inline` when the same class definition appears in a module to get inlining?

permalink embed save parent report reply

[–] **smdowney** 6 points 1 month ago

Yes, if you want the getters and setters inlined in the TU that imports them. Yes, this means a barrier to simple migration to modules. It was judged to be worth it for ABI control.

permalink embed save parent report reply

[–] **johannes1971** 6 points 1 month ago

That's unfortunate. After years of teaching people that the inline keyword is not for inlining but for ODR control we suddenly change direction, and the new direction requires them to make a judgement call that we just spent a decade teaching them they can't and shouldn't make themselves.

I don't quite understand why ABI control would be the reason, though. BMIs operate at the source level, before translation takes place, correct? Where does ABI come into it?

permalink embed save parent report reply

[–] **kalmoc** 6 points 1 month ago

This gives you explicit control over wether the body of your member function (and consequently everything used inside) becomes part of the ABI of your model or not.

Also, the meaning of inline hasn't really changed: "Just" the defaults did.

permalink embed save parent report reply

load more comments (1 reply)

[–] **abizjak3** 4 points 1 month ago

> In non-modules compilations, we deprecated cases where internal-linkage entities are used from external-linkage entities. (These cases typically lead to violations of the One Definition Rule.)

Can someone explain what that means? Using internal linkage entities from external-linkage entities is something I do very frequently, e.g. in a cpp file calling a function defined inline or in an unnamed namespace from an implementation of a function with external linkage.

permalink embed save report reply

[–] **zygoloid** C++ Project Editor | Clang Maintainer 8 points 1 month ago

Sorry for the imprecision here; it's hard to express detailed technical rules in a terse way. The thing that is deprecated is when the external linkage entity "exposes" the internal linkage entity -- either as part of its type, or as part of the body of an inline function, or similar. Non-inline definitions in .cpp files aren't affected.

permalink embed save parent report reply

load more comments (1 reply)

[–] **tpecholt** 3 points 1 month ago

Considering the new rules about ABI breakage what would be the chance for a new unordered_map proposal? If I understand correctly google has an implementation with order of magnitude better performance and same API

permalink embed save report reply

[–] **barchar** MSVC STL Dev 4 points 28 days ago

Well google's "flat_hash_map" is the one you usually want, and that's a different API than the std one (very different pointer invalidation grantees for one). IMHO it's probably more realistic to get the "flat" hashmap into the standard than it is to "fix" unordered_map. After all the flat hash map is a different type, with different tradeoffs.

permalink embed save parent report reply

load more comments (4 replies)

[–] **pjmlp** 8 points 1 month ago

Congratulations to everyone!

Even if I hardly use it nowadays, thanks for making C++ better.

permalink embed save report reply

[–] **kmhofmann** https://selene.dev 23 points 1 month ago

> Although there was strong interest in exploring how to evolve ABI in the future, we are not pursuing making C++23 a clean ABI breaking release at this time. We did, however, affirm that authors should be encouraged to bring individual papers for consideration, even if those would be an ABI break. Many in the committee are interested in considering targeted ABI breaks when that would signify significant performance gains.

That sounds like an utterly, utterly disappointing and meaningless conclusion of this discussion.

Good luck C++ with this committee - you might need it! </s>

permalink embed save report reply

[–] **sempuki** 23 points 1 month ago

This does not reflect reality. Until this vote, anything that proposed ABI was summarily dismissed. This will no longer be the case.

permalink embed save parent report reply

[–] **BrainIgnition** 17 points 1 month ago

Then maybe reword that paragraph? Currently it sounds like a divide and conquer strategy usually employed in politics: Generally agree with the solution at hand, but reject all concrete steps, because they're individually not worth it, etc. E.g.

> We declined to forward a paper to enhance std::regex to better support Unicode due to severe ABI restrictions

permalink embed save parent report reply

load more comments (12 replies)

[–] **kmhofmann** https://selene.dev 14 points 1 month ago

That's nice and all, but it's not enough, by far! The only sensible decision would have been to make a clean ABI break for C++23.
(In my opinion, ABI shouldn't even matter **at all** w.r.t. the C++ standard.)

With the wording above ("interested in considering"), I predict no actual ABI breaks to ever happen in practice.

permalink embed save parent report reply

load more comments (36 replies)

[–] **c0r3ntin** 12 points 1 month ago

It will be considered. And dismissed.

permalink embed save parent report reply

load more comments (5 replies)

load more comments (11 replies)

[–] **Ameisen** 3 points 1 month ago

How much time does it generally take you to write all that up?

permalink embed save report reply

[–] **bigcheesegs** Tooling Subgroup (SG15) Chair | Clang dev 11 points 1 month ago

We collaboratively edit this as a Google doc. This time it came together in 3 hours, but it's not like everyone was writing for that long.

permalink embed save parent report reply

load more comments (4 replies)

[–] **JulianHi93** 3 points 1 month ago

Does the current Reflection proposal define support for Reflection in a way that I'm able to generate new types?

permalink embed save report reply

[–] **andrewsutton** 8 points 1 month ago

That's in the source code injection proposals, which are in the pipeline.

permalink embed save parent report reply

load more comments (1 reply)

[–] **Morten242** 3 points 28 days ago

Apparently c++20 also adds a function called emit(), which breaks the "emit" keyword used in Qt.

A reasonable proposal[0] to rename the function to avoid this conflict was proposed but it was voted against with 20 strongly against. Is there anywhere to see the reason for why it was voted down so hard?

[0] https://cplusplus.github.io/LWG/issue3399

permalink embed save report reply

[–] **FabioFracassi** C++ Committee | Consultant 5 points 27 days ago

Because the situation where this can cause breakage are extremely rare (and can not happen in existing code, since the function is in a newly added header), and is almost trivial to work around should it arise.

Mitigation strategies exist within Qt (QT_NO_KEYWORDS), and using all lowercase macro names to define "keywords", is so far outside the agreed upon customs that I guess many were unwilling to cut Qt any slack there to (further) support such misuse.

permalink embed save parent report reply

load more comments (1 reply)

[–] tcanens cppreference.com | LWG 4 points 27 days ago

It was also not brought up until Friday, so the committee had less than 20 hours to decide before the 8pm deadline for straw polls. This magnifies the risk of making a change significantly.

permalink embed save parent report reply

[–] ezoe 13 points 1 month ago

It's sad that the C++ SC decided to taint std::format with locale. 10 years ago, I noticed the necessity of char8_t and they didn't listen, and now they not only think the locale is not considered harmful, but they also think locale helps localization. In reality, it's quite opposite, the locale actively hinder the localization effort.

Yes, Yes, its just type specifier n, but the problem is, it is implicitly available and anyone can use it innocently and it relies on the global locale of the time std::format object was initialized.

Well, it's not that bad. It just litter the standard library with yet another practically useless library after valarray, iostream and std::regex.

I also think the coroutines is ugly and it should better be handled by static reflection, if it got all the insane expressive power it aim to have currently that is.

permalink embed save report reply

[–] aearphen 6 points 1 month ago

Locales are supported via a very explicit opt-in and are occasionally useful e.g. for date/time formatting and inserting digit separators.

permalink embed save parent report reply

[–] mort96 3 points 1 month ago

How do you opt in? If it uses the global locale from `setlocale` at all, it's not opt-in, at least not when you're writing library code.

permalink embed save parent report reply

[–] PeterBrett SG16; CAD software dev 13 points 1 month ago

Locale is only ever used by `format` if you add a `L` modifier to your format substitution. There are overload that let you pass the specific locale to be used as a parameter. It's fully opt-in.

permalink embed save parent report reply

load more comments (9 replies)

[–] foonathan 6 points 1 month ago

> It's sad that the C++ SC decided to taint std::format with locale. 10 years ago, I noticed the necessity of char8_t and they didn't listen, and now they not only think the locale is not considered harmful, but they also think locale helps localization. In reality, it's quite opposite, the locale actively hinder the localization effort.

The committee does think locale is harmful. There are papers discussing alternatives in the Unicode study group.

> Yes, Yes, its just type specifier n, but the problem is, it is implicitly available and anyone can use it innocently and it relies on the global locale of the time std::format object was initialized.

There is also an overload where you pass in the locale as first parameter, instead of using the global one.

It's just convenience if you want to e.g. use , for floats.

permalink **embed** **save** **parent** **report** **reply**

[–] **FabioFracassi** C++ Committee | Consultant 9 points 1 month ago

`std::format` does *not* use locale by default... It uses it only if you explicitly provide one.

permalink **embed** **save** **parent** **report** **reply**

[–] **cpp_learner** 3 points 1 month ago

[formatter.requirements]:

> The output shall only depend on `t`, `fc.locale()`, and the range `[pc.begin(), pc.end())` from the last call to `f.parse(pc)`.

[format.context]:

> [`fc.locale()` returns] The locale passed to the formatting function if the latter takes one, and `std::locale()` otherwise.

IIUC, it means if a formatter decides to use locale, then it must default to `std::locale()`, a.k.a. the global locale.

permalink **embed** **save** **parent** **report** **reply**

[–] **aearphen** 4 points 1 month ago

It will use `std::locale()` or a locale passed to a formatting function but only if you request it via a separate format specifier. If you just do, say, `format("{:d}", 42)` locale won't be touched in any way.

permalink **embed** **save** **parent** **report** **reply**

load more comments (1 reply)

[–] **WafflesAreDangerous** 7 points 1 month ago

> It's just convenience if you want to e.g. use , for floats

Uhuh.. So the fact that Excels parsing of floating point numbers can in some contexts (opening CSV files for example) depend on if my locales decimal separator is fullstop or comma is a "convenicence". If this is the convenience you mean then I am .. ahem.. *impressed*.

permalink **embed** **save** **parent** **report** **reply**

load more comments (1 reply)

load more comments (1 reply)

[–] **TuxSH** 2 points 1 month ago

> We clarified the meaning of static (and unnamed namespaces)

Link is dead.

permalink **embed** **save** **report** **reply**

[–] **tcanens** cppreference.com | LWG 5 points 1 month ago

Not yet alive, rather. This is a paper that is revised at the meeting and will be in the post-meeting mailing.

permalink **embed** **save** **parent** **report** **reply**

load more comments (1 reply)

[–] [deleted] 1 month ago

[deleted]

[–] **tvaneerd** C++ Committee, lockfree, PostModernCpp 10 points 1 month ago

That is unlikely to happen. I haven't even heard of anyone suggesting that.

You should be able to turn warnings into errors via your compiler?

permalink **embed** **save** **report** **reply**

load more comments (1 reply)

load more comments (1 reply)

[–] **Suleyth** 2 points 1 month ago

So, well, now what's left for C++20 to come out? Like, to actually be usable in compilers and such?

permalink embed save report reply

[–] **FabioFracassi** C++ Committee | Consultant 9 points 1 month ago

Compilers and implementations have already started releasing c++20 features based on the working draft, and will continue to do so on their individual schedules... Concurrently ISO will do the balloting and red tape, and unless something mayorly unexpected happens, officially release the standard at the end of the year.

My totally unsubstantiated guess is that by that time we will have at least one implementation which is almost complete

permalink embed save parent report reply

[–] **kalmoc** 5 points 1 month ago

I would find that very suprising actually. There are so many big features in c++20 and if I'm not completely mistaken, no compiler implements even a single one of them completely to spec and in a production quality form.

permalink embed save parent report reply

[–] **[deleted]** 4 points 29 days ago

You are mistaken. https://en.cppreference.com/w/cpp/compiler_support

permalink embed save parent report reply

load more comments (6 replies)

[–] **barchar** MSVC STL Dev 6 points 28 days ago

Orc Peon voice: *work work*

permalink embed save parent report reply

load more comments (58 replies)