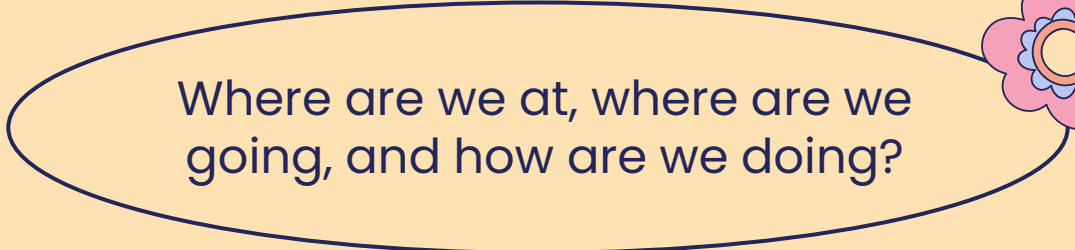




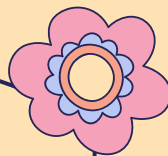

State of Clang as a C and C++ Compiler



with Aaron Ballman



Where are we at, where are we
going, and how are we doing?





01.



Retrospective

How have things been going over the past few years?





C++2c and C2y



Clang 17 started work on the upcoming C and C++ standards

- C++2c is expected in 2026
- C2y is expected sometime between 2026 and 2029

Both standards are expected to add significant new features, but neither has added a feature requiring major effort (yet).



Completed C++2c features

17 **User-generated static_assert messages**



= delete(“should have a reason”)



17 **constexpr cast from void ***



Attributes on structured bindings



18 **Placeholder variables with no name**



Unevaluated strings



19

19

18



Completed C++2c features

19

Trivial infinite loops are not UB



18

Remove deprecated arithmetic conversion on enumerations



19

Disallow binding a returned glvalue to a temporary



Pack indexing



19

Template parameter initialization



18

And more!



Completed C2y features

17

_Generic with a type operand



Always

++ and -- on _Complex values



Always

_Complex literals



Zero-length operations on null pointers



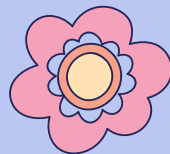
Always

typeof & typeof_unqual



Always





C++23 and C23



C++23 and C23 were both ratified by the standards committees in 2023 and are final, despite not being officially published by ISO

- Clang 19 is almost C++23 feature-complete, but lacks support for constexpr math, unknown pointers in constexpr, and explicit lifetime management
- Clang 19 implements significant support for C23, but lacks support for Decimal Floating Point and storage class specifiers for compound literals



Completed C++23 features

19

Deducing this



19

Portable assumptions



**Relaxing some constexpr
restrictions**

19



**constexpr needs to
propagate up**



And more!



17



Completed C23 features

19

constexpr for object definitions



18

Remove trigraphs??!



18

Type inference for object declarations (auto)



char8_t



19

Free positioning of labels



#embed



19

Completed C23 features

17

Revise spellings of keywords



17

Consistent initialization with }



17

nullptr and nullptr_t



17

Unreachable flow control



And more!





C++20 and C++17

C++20 was a major evolutionary change to C++, introducing major features like Modules, Concepts, and operator `<=>`,

- Clang 19 is almost C++20 feature complete, does not officially support modules

Thanks to significant efforts improving template sugaring and template template-argument matching, Clang 19 is now C++17 feature complete



Completed C++20 features



19

Concepts



Lambdas in unevaluated contexts



17

17

CTAD for alias aggregates



Generalized NTTP of scalar types



18

17

operator \Leftrightarrow




constexpr



17





What's going on with Modules?

Modules were added to C++20, work started on them in Clang 11

- Support is partial in Clang 19, feature test macro is not defined
- **However, C++ modules are well-supported in Clang 19 and should work for common usage patterns**
- Still to be done (Issue #112295):
 - Diagnostics for TU-local entities
 - Module-level lookup
- Community is continuing to work towards full support

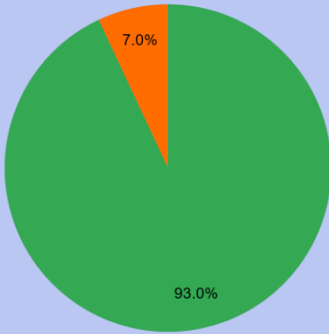


C conformance by the numbers

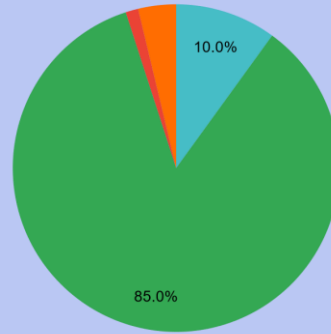
as of Oct 15, 2024



C99 Support

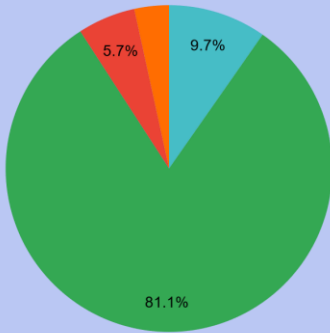


C11 Support

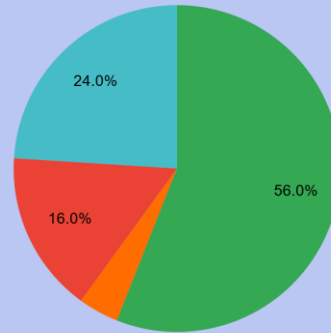


- Unknown
- Yes
- No
- Partial

C23 Support



C2y Support

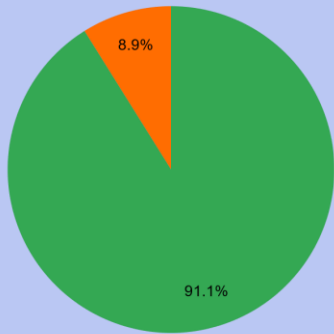


C++ conformance by the numbers

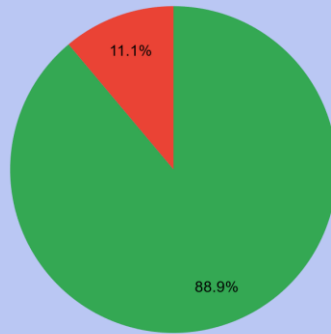
as of Oct 15, 2024



C++20 Support

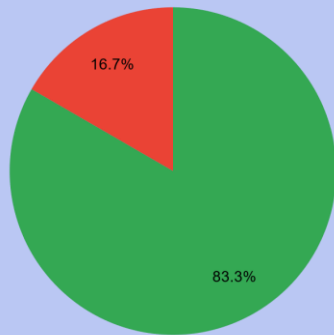


C++23 Support



● Yes
● No

C++2c Support



Clang is fully conforming to C++11, C++14, and C++17

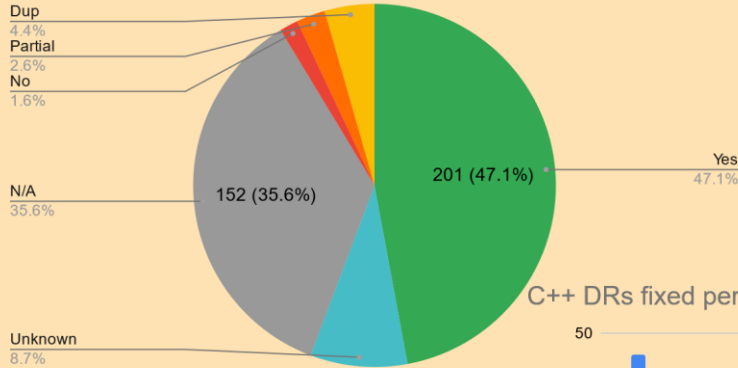


Defect reports by the numbers

as of Oct 15, 2024

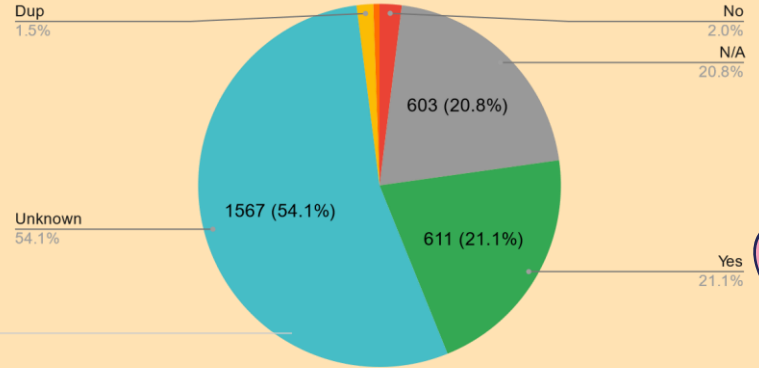
C DR Support

Total: 427 DRs

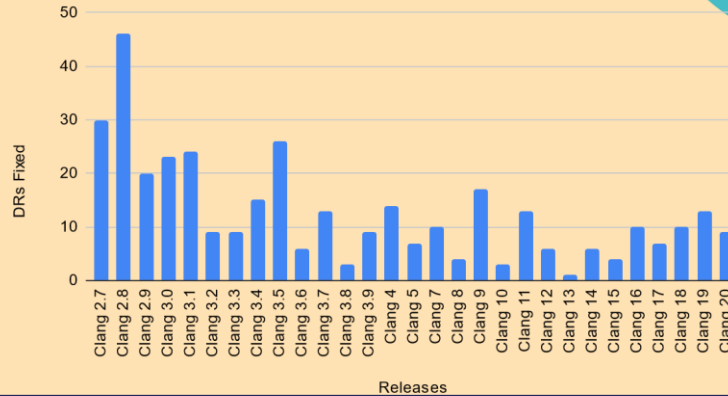


C++ DR Support

Total: 2897 DRs

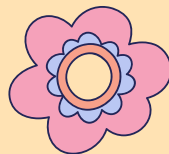


C++ DRs fixed per release





02.



How 'bout now, brown cow?

What are we up to today?



Current C++ Efforts

C++20

- Variadic friends
- constexpr placement new
- Reflection
- Contracts

C++20

- Modules!
- Defect reports

C++23

- Lifetime extension in range-based for loops
- Unknown pointers and references in constant expressions
- CTAD for inherited constructors
- constexpr math

Extensions

- SYCL, OpenACC
- Analyses for real-time code





Current C Efforts

C2y

- Testing existing extensions that were standardized
- New features (`_Lengthof`, `'if'` declarations, named loops, etc)

C23

- Improved normal enumerations
- Enumerations with a fixed underlying type
- Decimal floating-point support

Extensions

- Bounds safety checking!
- Connecting flexible array members to their count

Other

- Hardening diagnostics
- Conformance testing





03.



Word on the street

What are people saying about Clang?



Performance matters

Runtime Performance

- Clang and GCC are competitive with each other
- Clang and MSVC are competitive with each other
- Vector math optimizes very well, even in debug mode

Compile Time Performance

- Compile times are getting slower with mixed results for runtime
- Folks are using -ftime-trace and are still not successful
- It's not just Clang; C++ features are adding significant overhead and that impacts all vendors

Tooling matters

clang-tidy

- Seeing increased use of clang-tidy, especially in precommit CI
- Folks are integrating it in CMake scripts as part of their build process

clang-format

- Being integrated into many different tools (MSVC, vim, emacs, Clion, ReSharper, etc)
- Number of options can be overwhelming, but users appreciate the flexibility
- Also has a lot of precommit CI integration

clangd


- Very popular, especially with VS Code users, but is flexible enough to be used with almost anything





Standards committees matter

WG14 (C) and WG21 (C++)

- C committee is happy with Clang's ongoing implementation of C23
 - C++ committee is happy with Clang, some concerns about speed of implementing newer C++ standards
 - Both committees have some members concerned that compiler extensions eat into committee design space
 - There's no good mechanism for community collaboration with the standards committees on feature design
- 

Overall external perception

Good Stuff

- Users are excited by some of our extensions (a lot of love for `-fbounds-safety`)
- Users appreciate the full suite of offerings from Clang (compiler, static analyzer, tooling, etc)
- We are competitive in terms of features and performance
- We still have very good diagnostic messages and standards conformance
- Users appreciate Clang as an alternative to GCC and MSVC

Less Good Stuff

- People notice that we're not fully conforming to the latest standards
- There's some frustration about compile time overhead
- Our documentation needs love
- Perception is that corporate interests dictate project direction rather than being a "real" OSS project



04.



Wrapping it up

My thoughts on how things are going





Kudos!

We've made serious changes which have improved the user experience:

- Clang 13/14 release notes were almost empty; users noticed, we reacted, and Clang 17/18/19 each have hundreds of release notes
- We're improving how we interact on issues; a little bit less like screaming into the void now
- We've grown! More people helping with code reviews, issue triage, answering questions, etc than ever before!



What can we improve?



Docs

Especially around implementation-defined behaviors and extensions



Testing

We can't buy a test suite, so what do we do?



Issues

We need to improve on reacting to issues, especially regressions



Growth

How do we grow the community, especially in non-coding areas?



**No, but
seriously,
Kudos!**



We wouldn't have anything to offer anyone if it wasn't for the many, many volunteers who make Clang what it is. Thank you!





THANKS!

Do you have any questions?

Email:

aaron@aaronballman.com

Discord/IRC/GitHub/Discourse:

AaronBallman

CREDITS: This presentation template was created by Slidesgo, including icons by Flaticon, infographics & images by Freepik

