C++17

C++17 is a revision of the ISO/IEC 14882 standard for the C++ programming language.

Contents

History

Before the C++ Standards Committee fixed a 3-year release cycle, C++17's release date was uncertain. In that time period, the C++17 revision was also called **C++1z**, following C++0x or C++1x for C++11 and C++1y for C++14. The C++17 specification reached the Draft International Standard (DIS) stage in March 2017.[1][2] This DIS was unanimously approved, with only editorial comments,[3] and the final standard was published in December 2017.[4] Few changes were made to the C++ Standard Template Library, although some algorithms in the `<algorithm>` header were given support for explicit parallelization and some syntactic enhancements were made.

Removed

This revision of C++ not only added new features but also removed a few.

- Removal of trigraphs.[5][6]
- Removal of some deprecated types and functions from the standard library, including `std::auto_ptr`, `std::random_shuffle`, and old function adaptors.[7][8] These were superseded in C++11 by improved facilities such as `std::unique_ptr`, `std::shuffle`, `std::bind`, and lambdas.
- Removal of the (formerly deprecated) use of the keyword `register` as a storage class specifier.[9] This keyword is now reserved and unused.

New features

C++17 introduced many new features. The following lists may be incomplete.

Language

- Making the text message for `static_assert` optional[10]
- Allow `typename` (as an alternative to `class`) in a template template parameter[11]
- New rules for `auto` deduction from braced-init-list[12][7]
- Nested namespace definitions, e.g., `namespace X::Y { … }` instead of `namespace X { namespace Y { … } }`[7][13]
- Allowing attributes for namespaces and enumerators[14][15]
- New standard attributes `[[fallthrough]]`, `[[maybe_unused]]` and `[[nodiscard]]`[16]
- UTF-8 (u8) character literals[14][17] (UTF-8 string literals have existed since C++11; C++17 adds the corresponding character literals for consistency, though as they are restricted to a single byte they can only store ASCII)
- Hexadecimal floating-point literals[18][19]
- Use of `auto` as the type for a non-type template parameter[20]
- Constant evaluation for all non-type template arguments[14][21]
- Fold expressions, for variadic templates[14][22]
- A compile-time static `if` with the form `if constexpr(expression)`[23]
- Structured binding declarations, allowing `auto [a, b] = getTwoReturnValues();`[24]
- Initializers in `if` and `switch` statements[25]
- copy-initialization and direct-initialization of objects of type `T` from prvalue expressions of type `T` (ignoring top-level cv-qualifiers) shall result in no copy or move constructors from the prvalue expression. See copy elision for more information.
- Some extensions on over-aligned memory allocation[26]
- Class template argument deduction (CTAD), introducing constructor deduction guides, eg. allowing `std::pair(5.0, false)` instead of requiring explicit constructor arguments types `std::pair<double, bool>(5.0, false)` or an additional helper template function `std::make_pair(5.0, false)`.[27][28]
- Inline variables, which allows the definition of variables in header files without violating the one definition rule. The rules are effectively the same as inline functions
- `__has_include`, allowing the availability of a header to be checked by preprocessor directives[29]
- Value of `__cplusplus` changed to `201703L`[30]
- Exception specifications were made part of the function type[31]

Library

- Most of Library Fundamentals TS I, including:[32][33]
  - `std::string_view`, a read-only non-owning reference to a character sequence or string-slice[34]
  - `std::optional`, for representing optional objects, a data type that may not always be returned by a given algorithm with support for non-return
  - `std::any`, for holding single values of any type
- `std::uncaught_exceptions`, as a replacement of `std::uncaught_exception` in exception handling[35][14]

- New insertion functions `try_emplace` and `insert_or_assign` for `std::map` and `std::unordered_map` key-value associative data structures[36][37]
- Uniform container access: `std::size`, `std::empty` and `std::data`[37][38]
- Definition of "contiguous iterators"[37][39]
- A file system library based on `boost::filesystem`[40]
- Parallel versions of STL algorithms[41]
- Additional mathematical special functions, including elliptic integrals and Bessel functions[42]
- `std::variant`, a tagged union container[43]
- `std::byte`, allowing char to be replaced for data types intending to model a byte of data as a byte rather than a character[44]
- Logical operator traits: `std::conjunction`, `std::disjunction` and `std::negation`[45]
- `<memory_resource>` header, for polymorphic memory resources[46]

Compiler support

- GCC has had complete support for C++17 language features since version 8. [47]
- Clang 5 and later implement all the features of C++17.[48]
- Visual Studio 2017 15.8 (MSVC 19.15) supports all of C++17.[49][50]

Library support

- libstdc++ since version 9.1 has complete support for c++17 (8.1 without Parallelism TS and referring to C99 instead of C11) [51]
- libc++ as of version 9 has partial support for c++17, with the remainder "in progress" [52]
- MSVC Standard Library since 19.15 complete support for c++17 except for "Elementary String Conversions" and referring to C99 instead of C11.[53]

See also

- C++ compilers
- C11 (C standard revision)
- C17 (C standard revision)

References

1. "N4661 Editors' Report -- Programming Languages -- C++" (http://www.open-std.org/jtc1/sc22/wg21/docs/papers/2017/n4661.html). 21 March 2017. Retrieved 2017-03-21.
2. "ISO/IEC DIS 14882: Programming Languages — C++" (https://web.archive.org/web/20170325025026/http://www.open-std.org/jtc1/sc22/wg21/docs/papers/2017/n4660.pdf) (PDF). Archived from the original (http://www.open-std.org/jtc1/sc22/wg21/docs/papers/2017/n4660.pdf) (PDF) on 2017-03-25.
3. Herb Sutter. "C++17 is formally approved" (https://herbsutter.com/2017/09/06/c17-is-formally-approved/).
4. "ISO/IEC 14882:2017" (https://www.iso.org/standard/68564.html).

5. "N3981: Removing trigraphs??! (Richard Smith)" (http://www.open-std.org/JTC1/SC22/WG21/docs/papers/2014/n3981.html). 2014-05-06.
6. IBM comment on preparing for a Trigraph-adverse future in C++17 (http://www.open-std.org/jtc1/sc22/wg21/docs/papers/2014/n4210.pdf), IBM paper N4210, 2014-10-10. Authors: Michael Wong, Hubert Tong, Rajan Bhakta, Derek Inglis
7. "Updates to my trip report" (http://isocpp.org/blog/2014/11/updates-to-my-trip-report).
8. "N4190: Removing auto_ptr, random_shuffle(), And Old <functional> Stuff (Stephan T. Lavavej)" (http://www.open-std.org/jtc1/sc22/wg21/docs/papers/2014/n4190.htm).
9. "C++ Keywords: register" (https://en.cppreference.com/w/cpp/keyword/register).
10. "N3928: Extending static_assert, v2 (Walter E. Brown)" (http://www.open-std.org/jtc1/sc22/wg21/docs/papers/2014/n3928.pdf) (PDF).
11. "N4051: Allow typename in a template template parameter (Richard Smith)" (http://www.open-std.org/jtc1/sc22/wg21/docs/papers/2014/n4051.html).
12. "N3922: New Rules for auto deduction from braced-init-list (James Dennett)" (http://www.open-std.org/jtc1/sc22/wg21/docs/papers/2014/n3922.html).
13. "N4230: Nested namespace definition (Robert Kawulak, Andrew Tomazos)" (http://www.open-std.org/jtc1/sc22/wg21/docs/papers/2014/n4230.html).
14. "New core language papers adopted for C++17" (https://isocpp.org/blog/2014/11/new-papers-adopted-for-cpp17).
15. "N4266: Attributes for namespaces and enumerators (Richard Smith)" (http://isocpp.org/files/papers/n4266.html).
16. "N4640: Working Draft, Standard for Programming Language C++" (http://www.open-std.org/jtc1/sc22/wg21/docs/papers/2017/n4640.pdf) (PDF). pp. 193–195.
17. "N4267: Adding u8 character literals (Richard Smith)" (http://isocpp.org/files/papers/n4267.html).
18. Thomas Köppe. "Hexadecimal floating literals for C++" (http://wg21.link/p0245r1).
19. "N4659: Working Draft, Standard for Programming Language C++" (http://www.open-std.org/jtc1/sc22/wg21/docs/papers/2017/n4659.pdf) (PDF). §5.13.4.
20. James Touton; Mike Spertus (2016-06-23). "Declaring non-type template parameters with auto" (http://www.open-std.org/jtc1/sc22/wg21/docs/papers/2016/p0127r2.html).
21. "N4268: Allow constant evaluation for all non-type template arguments (Richard Smith)" (http://isocpp.org/files/papers/n4268.html).
22. "N4295: Folding expressions (Andrew Sutton, Richard Smith)" (http://isocpp.org/files/papers/n4295.html).
23. "N4659: Working Draft, Standard for Programming Language C++" (http://www.open-std.org/jtc1/sc22/wg21/docs/papers/2017/n4659.pdf) (PDF). §9.4.1.
24. "N4659: Working Draft, Standard for Programming Language C++" (http://www.open-std.org/jtc1/sc22/wg21/docs/papers/2017/n4659.pdf) (PDF). §11.5.
25. "Selection statements with initializer" (http://www.open-std.org/jtc1/sc22/wg21/docs/papers/2016/p0305r1.html).
26. "Dynamic memory allocation for over-aligned data" (http://www.open-std.org/jtc1/sc22/wg21/docs/papers/2016/p0035r4.html).
27. "Class template argument deduction" (https://en.cppreference.com/w/cpp/language/class_template_argument_deduction).
28. "CppCon 2018: Timur Doumler "Class template argument deduction in C++17" " (https://www.youtube.com/watch?v=UDs90b0yjjQ).
29. "N4640: Working Draft, Standard for Programming Language C++" (http://www.open-std.org/jtc1/sc22/wg21/docs/papers/2017/n4640.pdf) (PDF). pp. 431–433.

30. "N4659: Working Draft, Standard for Programming Language C++" (http://www.open-std.org/jtc1/sc22/wg21/docs/papers/2017/n4659.pdf) (PDF). §19.8.
31. "P0012R1: Make exception specifications be part of the type system, version 5" (http://www.open-std.org/jtc1/sc22/wg21/docs/papers/2015/p0012r1.html).
32. "Adopt Library Fundamentals V1 TS Components for C++17 (R1)" (https://isocpp.org/files/papers/p0220r1.html).
33. "Current Status" (https://isocpp.org/std/status).
34. "std::basic_string_view - cppreference.com" (http://en.cppreference.com/w/cpp/string/basic_string_view). *en.cppreference.com*. Retrieved 2016-06-23.
35. "N4259: Wording for std::uncaught_exceptions (Herb Sutter)" (http://isocpp.org/files/papers/n4259.pdf) (PDF).
36. "N4279: Improved insertion interface for unique-key maps (Thomas Köppe)" (https://isocpp.org/files/papers/n4279.html).
37. "New standard library papers adopted for C++17" (https://isocpp.org/blog/2014/11/new-standard-library-papers-adopted-for-cpp17).
38. "N4280: Non-member size() and more (Riccardo Marcangelo)" (https://isocpp.org/files/papers/n4280.pdf) (PDF).
39. "N4284: Contiguous Iterators (Jens Maurer)" (https://isocpp.org/files/papers/n4284.html).
40. "Filesystem Library Proposal (Beman Dawes)" (http://www.open-std.org/jtc1/sc22/wg21/docs/papers/2013/n3505.html).
41. "The Parallelism TS Should be Standardized" (https://isocpp.org/files/papers/P0024R2.html).
42. "Mathematical Special Functions for C++17, v5" (https://isocpp.org/files/papers/P0226R1.pdf) (PDF).
43. "N4659: Working Draft, Standard for Programming Language C++" (http://www.open-std.org/jtc1/sc22/wg21/docs/papers/2017/n4659.pdf) (PDF). §23.7.
44. "A byte type definition" (http://open-std.org/JTC1/SC22/WG21/docs/papers/2017/p0298r3.pdf) (PDF).
45. "N4659: Working Draft, Standard for Programming Language C++" (http://www.open-std.org/jtc1/sc22/wg21/docs/papers/2017/n4659.pdf) (PDF). §23.15.8.
46. "PMR (Polymorphic Memory Resources) fully described -- Nico Josuttis" (https://isocpp.org/blog/2018/10/pmr-polymorphic-memory-resources).
47. "C++ Standards Support in GCC - GNU Project - Free Software Foundation (FSF)" (https://gcc.gnu.org/projects/cxx-status.html). *gcc.gnu.org*.
48. "Clang - C++17, C++14, C++11 and C++98 Status" (https://clang.llvm.org/cxx_status.html). *clang.llvm.org*.
49. corob-msft. "Visual C++ Language Conformance" (https://docs.microsoft.com/en-us/cpp/visual-cpp-language-conformance). *docs.microsoft.com*.
50. "Announcing: MSVC Conforms to the C++ Standard" (https://blogs.msdn.microsoft.com/vcblog/2018/05/07/announcing-msvc-conforms-to-the-c-standard/).
51. "Chapter 1. Status" (https://gcc.gnu.org/onlinedocs/libstdc++/manual/status.html). *gcc.gnu.org*.
52. "libc++ C++17 Status" (http://libcxx.llvm.org/cxx1z_status.html). *llvm.org*.
53. "Announcing: MSVC Conforms to the C++ Standard" (https://devblogs.microsoft.com/cppblog/announcing-msvc-conforms-to-the-c-standard/). *devblogs.microsoft.com*.
