DRAFT INTERNATIONAL STANDARD

ISO/IEC DIS 14882

ISO/IEC JTC **1**/SC **22** Secretariat: **ANSI**

Voting begins on:
2020-06-11

Voting terminates on:
2020-09-03

Programming languages — C++

Langages de programmation — C++

ICS: 35.060

This document is circulated as received from the committee secretariat.

Reference number
ISO/IEC DIS 14882:2020(E)

COPYRIGHT PROTECTED DOCUMENT

ISO copyright office
CP 401 • Ch. de Blandonnet 8
CH-1214 Vernier, Geneva
Phone: +41 22 749 01 11
Fax: +41 22 749 09 47
Email: copyright@iso.org
Website: www.iso.org

Published in Switzerland

Contents

Foreword

ISO (the International Organization for Standardization) and IEC (the International Electrotechnical Commission) form the specialized system for worldwide standardization. National bodies that are members of ISO or IEC participate in the development of International Standards through technical committees established by the respective organization to deal with particular fields of technical activity. ISO and IEC technical committees collaborate in fields of mutual interest. Other international organizations, governmental and non-governmental, in liaison with ISO and IEC, also take part in the work.

The procedures used to develop this document and those intended for its further maintenance are described in the ISO/IEC Directives, Part 1. In particular the different approval criteria needed for the different types of documents should be noted. This document was drafted in accordance with the editorial rules of the ISO/IEC Directives, Part 2 (see `www.iso.org/directives`).

Attention is drawn to the possibility that some of the elements of this document may be the subject of patent rights. ISO and IEC shall not be held responsible for identifying any or all such patent rights. Details of any patent rights identified during the development of the document will be in the Introduction and/or on the ISO list of patent declarations received (see `www.iso.org/patents`) or the IEC list of patent declarations received (see `http://patents.iec.ch`).

Any trade name used in this document is information given for the convenience of users and does not constitute an endorsement.

For an explanation on the voluntary nature of standards, the meaning of ISO specific terms and expressions related to conformity assessment, as well as information about ISO's adherence to the World Trade Organization (WTO) principles in the Technical Barriers to Trade (TBT) see the following URL: `www.iso.org/iso/foreword.html`.

This document was prepared by Joint Technical Committee ISO/IEC JTC 1, *Information technology*, Subcommittee SC 22, *Programming languages, their environments and system software interfaces*.

This sixth edition cancels and replaces the fifth edition (ISO/IEC 14882:2017), which has been technically revised. It also incorporates the Technical Specifications

- ISO/IEC TS 19217:2015 *Programming Languages — C++ Extensions for concepts*,
- ISO/IEC TS 21425:2017 *Programming Languages — C++ Extensions for ranges*,
- ISO/IEC TS 22277:2017 *Programming Languages — C++ Extensions for Coroutines*,
- ISO/IEC TS 21544:2018 *Programming Languages — Extensions to C++ for Modules*,
- portions of ISO/IEC TS 19571:2016 *Programming Languages — Technical specification for C++ extensions for concurrency*, and
- portions of ISO/IEC TS 19568:2017 *Programming Languages — C++ Extensions for Library Fundamentals*.

The main changes compared to the previous edition are as follows:

- addition of concepts, *requires-clause*s, and *requires-expression*s and `<concepts>` (18.3) header
- addition of coroutines, including `co_yield`, `co_await`, and `co_return` keywords and `<coroutine>` (17.12.1) header
- addition of modules, *import-declaration*s, and *export-declaration*s
- addition of three-way comparison, defaulted comparisons, rewriting of comparison operator expressions, and `<compare>` (17.11.1) header
- addition of designated initializers
- support for class types and floating-point types as the type of a non-type template parameter
- new attributes `[[no_unique_address]]`, `[[likely]]`, `[[unlikely]]`
- support for optional reason string in `[[nodiscard]]` attribute
- ability to require constant initialization with `constinit` keyword

- ability to require constant evaluation with `consteval` keyword
- extension of constant evaluation support to cover
  - memory allocation and deallocation
  - virtual function calls
  - `dynamic_cast` and `typeid`
  - changing the active member of a union
  - uninitialized variables
- support for constexpr functions to contain `catch` handlers and *asm-declaration*s that are not reached during constant evaluation
- support for controlling destruction in a class-specific operator delete function
- addition of `using enum` declaration
- addition of `char8_t` type
- guarantee that `char16_t` and `char32_t` literals are encoded as UTF-16 and UTF-32 respectively
- support for an initializer statement in range-based for loops
- support for default member initializers for bit-fields
- support for parenthesized aggregate initialization
- extended support for lambda expressions, including
  - explicit *template-head*s in generic lambdas
  - default construction and assignment of stateless closure types
  - lambda expressions in unevaluated operands
  - pack expansion of lambda *init-captures*
- generalized support for structured bindings
- support for inline namespaces in nested namespace definitions
- support for conditionally-explicit member functions
- extended support for class template argument deduction to cover aggregate initialization and alias templates
- reduced cases in which `typename` is required
- support for calling an undeclared *template-id* via argument-dependent name lookup
- relaxed access checking rules in template specialization declarations
- revised memory model
- expanded cases in which returned or thrown variables are implicitly moved
- extended support for variadic macros with `__VA_OPT__`
- feature test macros and `<version>` (17.3.2) header
- restricted valid uses of standard library functions and function templates
- addition of ranges and `<ranges>` (24.2) header
- addition of calendar and time zone support
- addition of text formatting library and `<format>` (20.20.1) header
- addition of synchronization facilities for waiting, notifications, semaphores, latches, and barriers, and `<barrier>` (32.8.2.1), `<latch>` (32.8.1.1) and `<semaphore>` (32.7.1) headers
- addition of mathematical constants library and `<numbers>` (26.9.1) header
- support for representing source locations and `<source_location>` (17.8.1) header
- addition of `span` view and `<span>` (22.7.2) header
- addition of joining thread class and `<stop_token>` (32.3.2) header
- extensions to atomic types and operations, including

  - new class template `atomic_ref`
  - new atomic floating-point operations
  - new atomic smart pointer types
  - support for compare-and-exchange operations on types with padding
  - conversion of `memory_order` to a scoped enumeration
- addition of `unsequenced` execution policy
- new utility functions, types, and templates in the standard library, including
  - new type traits `is_bounded_array`, `is_corresponding_member`, `is_layout_compatible`, `is_nothrow_convertible`, `is_pointer_interconvertible_base_of`, `is_pointer_interconvertible_with_class`, and `is_unbounded_array`
  - new transformation traits `remove_cvref`, `type_identity`, `unwrap_ref_decay`, and `unwrap_reference`
  - new standard library primitive functions `assume_aligned` and `is_constant_evaluated`
  - new free functions `erase` and `erase_if` for containers
  - utilities for uses-allocator construction
  - function template `to_address`
  - function template `bind_front`
  - function template `to_array`
- `make_shared` support for array types
- support for allocating objects owned by `shared_ptr` and `unique_ptr` with default initialization
- addition of bit manipulation library and `<bit>` (26.5.2) header
- addition of a synchronized buffered output stream and `<syncstream>` (29.10.1) header
- addition of `shift` algorithms
- addition of `midpoint` and `lerp` math functions
- use of `[[nodiscard]]` attribute in the standard library
- support for heterogeneous lookup for unordered containers
- support for element existence detection in associative containers
- change to return removed element count from `remove`, `remove_if`, and `unique` member functions on `list` and `forward_list`
- addition of `starts_with` and `ends_with` to `basic_string` and `basic_string_view`
- support for move semantics in `<numeric>` (25.9) algorithms
- support for efficient access to the buffer of a `basic_stringbuf`
- extended constant expression evaluation support in the standard library to cover more algorithms, utilities, and types, including `pair`, `tuple`, `vector`, and `string`
- removal of deprecated features

Any feedback or questions on this document should be directed to the user's national standards body. A complete listing of these bodies can be found at www.iso.org/members.html.

1 Scope [intro.scope]

1 This document specifies requirements for implementations of the C++ programming language. The first such requirement is that they implement the language, so this document also defines C++. Other requirements and relaxations of the first requirement appear at various places within this document.

2 C++ is a general purpose programming language based on the C programming language as described in ISO/IEC 9899:2018 *Programming languages — C* (hereinafter referred to as the *C standard*). C++ provides many facilities beyond those provided by C, including additional data types, classes, templates, exceptions, namespaces, operator overloading, function name overloading, references, free store management operators, and additional library facilities.

