Advanced programming in C++

The plan

- C++ history and development (C++ Committee)
- Present and close future C++ standard (C++11/14/17)
- Valid C++ code organization
- Memory management in C++
- Exceptions
- Namespaces
- (Advanced) templates
- Generic programming
- Design Patterns
- The Standard Library (STL)
  - C Standard Library
  - Containers
  - Threads
  - IO
  - Algorithms
  - Etc.
- Metaprogramming
- Boost library

History of C++

- C
  - 1972 – C language (pointers, array-pointer, ++, structs)
  - 1973 – UNIX rewritten with C
  - 1978: The C Programming Language, 1st edition
  - …
- 1979: C with Classes first implemented

  New features: classes, member functions, derived classes, separate compilation, public and private access control, friends, type checking of function arguments, default arguments, inline functions, overloaded assignment operator, constructors, destructors, f() same as f(void), call-function and return-function
- 1985: Cfront 1.0 (C++ to C converting compiler)

  New features: virtual functions, function and operator overloading, references, new and delete operators, the keyword const, scope resolution operator
- 1985: The C++ Programming Language, 1st edition
- 1987: C++ support in GCC 1.15.3

History of C++

- 1989: Cfront 2.0

  New features: multiple inheritance, pointers to members, protectedccess, type-safe linkage, abstract classes, static and const member functions, class-specific new and delete

- 1991: The C++ Programming Language, 2nd edition
- 1990 ANSI C++ Committee founded
- 1991 ISO C++ Committee founded
- 1992 STL implemented in C++
- 1998 C++98 (ISO/IEC 14882:1998)

  New features: RTTI (dynamic_cast, typeid), covariant return types, cast operators, mutable, bool, declarations in conditions, template instantiations, member templates, export

  Library additions: containers, algorithms, iterators, function objects (based on STL), locales, bitset, valarray, auto_ptr, templatized string, iostream, and complex.

- 1998 The C++ Programming Language, 3rd edition

History of C++

- 1999 Boost founded by the committee members to produce new high-quality candidate libraries for the standard.
- 2003 C++03 (ISO/IEC 14882:2003)
- 2006 Performance Technical Report (ISO/IEC TR 18015:2006) (ISO Store) (2006 draft)

  This TR discussed the costs of various C++ abstractions, provided implementation guidance, discussed use of C++ in embedded systems and introduced <hardware> interface to C's ISO/IEC TR 18037:2008 <iohw.h>.
- 2007 Library extension TR1 (ISO/IEC TR 19768:2007) (ISO store) (2005 draft).

  This TR is a C++ library extension, which adds from Boost to the C++ standard library: Reference wrapper, Smart pointers, Member function, Result Of, Bind, Function, Type Traits, Random, Mathematical Special Functions, Tuple, Array, Unordered Containers (including Hash), and Regular Expressions.
- **2011 C++11 (ISO/IEC 14882:2011)**

2011 C++11 (ISO/IEC 14882:2011)

- A large number of changes were introduced to both standardize existing practices and improve the abstractions available to the C++ programmers
- New language features: auto and decltype, defaulted and deleted functions, final and override, trailing return type, rvalue references, move constructors/move assignment, scoped enums, constexpr and literal types, list initialization, delegating and inherited constructors, brace-or-equal initializers, nullptr, long long, char16_t and char32_t, type aliases, variadic templates, generalized unions, generalized PODs, Unicode string literals, user-defined literals, attributes, lambda expressions, noexcept, alignof and alignas, multithreaded memory model, thread-local storage, GC interface, range for (based on a Boost library), static assertions (based on a Boost library)
- New library features: atomic operations library, emplace() and other use of rvalue references throughout all parts of the existing library, std::initializer_list, stateful and scoped allocators, forward_list, chrono library, ratio library, new algorithms, Unicode conversion facets
- From TR1: (almost) all of TR1
- From Boost: The thread library, exception_ptr, error_code and error_condition, iterator improvements (std::begin, std::end, std::next, std::prev)
- From C: C-style Unicode conversion functions

History of C++

- 2013 The C++ Programming Language, 4th edition
- 2014 C++14

  Minor revision of the C++ standard

  - New language features: variable templates, polymorphic lambdas, lambda captures expressions, new/delete elision, relaxed restrictions on constexpr functions, binary literals, digit separators, return type deduction for functions, aggregate initialization for classes with brace-or-equal initializers.
  - New library features: std::make_unique, std::shared_timed_mutex and std::shared_lock, std::integer_sequence, std::exchange, std::quoted, and many small improvements to existing library facilities, such as two-range overloads for some algorithms, type alias versions of type traits, user-defined string, duration, and complex number literals, etc.

History of C++

- 2015 Filesystem library Technical Specification
  - An experimental C++ library extension that specifies a filesystem library based on boost.filesystem V3 (with some modifications and extensions). Merged into C++17.
- 2015 Extensions for Parallelism TS
  - Standardizes parallel and vector-parallel API for all standard library algorithms, as well as adds new algorithms such as reduce, transform_reduce, or exclusive_scan. Merged into C++17.
- 2015 Extensions for Transactional Memory TS
  - Extends the C++ core language with synchronized and atomic blocks, as well as transaction-safe functions, which implement transactional memory semantics.
- 2015 Extensions for Library Fundamentals TS
  - Adds several new components to the C++ standard library: optional, any, string_view, sample, search, apply, polymorphic allocators, and variable templates for type traits. Merged into C++17.
- 2015 Extensions for Concepts TS
  - Extends the C++ core language with concepts (named type requirements) and constraints (limits on the types allowed in template, function, and variable declarations), which aids metaprogramming and simplifies template instantiation diagnostics, see concepts
- 2016 Extensions for Concurrency TS
  - Extends the C++ library to include several extensions to std::future, latches and barriers, and atomic smart pointers.

Future developments

- Experimental technical specifications (Networking, Ranges, Modules, Numerics, versions 2 of Library Extensions, Parallelism, Concurrency)
- 2017 C++17
  - Language changes: fold-expressions, class template deduction (std::vector v = {1,2,3};), auto non-type template parameters (template <auto v> struct B), compile-time if constexpr, inline variables, decomposition declaration (auto [a,b] = f();), initializers for if and switch if(init;condition) and switch(init;condition), u8-char, simplified nested namespaces, using-declaration can declare multiple names, made noexcept part of type system, allocation functions with explicit alignment, guaranteed copy elision, __has_include, lambda capture of *this, constexpr lambda, attribute namespaces don't have to repeat, new attributes [[fallthrough]], [[nodiscard]], and [[maybe_unused]]. Removed trigraphs, the register keyword, bool increment
  - Library changes: std::variant, std::has_unique_object_representations, std::launder, uninitialized memory tools (std::destroy_at/std::destroy/std::destroy_n, std::uninitialized_move, std::uninitialized_value_construct, etc), map/set extract and map/set merge, std::make_from_tuple, std::to_chars/from_chars, relative path support in Filesystem (proximate, relative, weakly_canonical), contiguous iterators, non-member size/empty/data, map/unordered_map try_emplace and insert_or_assign, uncaught_exceptions, variadic lock_guard, as_const, conjunction/disjunction/negation, type trait variables xxx_v, transparent owner_less, rounding functions for duration and time_point, array support for std::shared_ptr, not_fn, weak_from_this, is_always_lock_free, is_swappable, clamp, 3D hypot, cache line interface, is_callable. Deprecated std::iterator, std::raw_storage_iterator, get_temporary_buffer, is_literal_type, Removed auto_ptr, the deprecated function objects, the obsolete iostreams aliases.
  - From TS's: the filesystem library, the library fundamentals v1 (including optional, any, string_view, polymorphic allocators, searchers, apply), parallelism v1 (including execution policies, reduce, inclusive_scan, exclusive_scan), but removing exception_list. From special math IS: special mathematical functions, from library fundamentals v2: std::gcd, std::lcm
  - From C11: std::aligned_alloc, std::timespec_get
- 2020 C++20 ???

The Committee: WG21

- The ISO C++ committee, officially ISO/IEC JTC1 (Joint Technical Committee 1) / SC22 (Subcommittee 22) / WG21 (Working Group 21).
- The WG21 officers are:
  - Convener: Herb Sutter (Microsoft)
  - Project Editor: Richard Smith (Google)
  - Secretary: Jonathan Wakely (Red Hat).
- The U.S. national C++ committee (officially ANSI X3J16; now called .16)
  - Chair: Clark Nelson (Intel)
  - Vice-Chair: John Spicer (Edison Design Group – C++ tools included by Intel, Microsoft, etc.)
  - Secretary: Jonathan Wakely (Red Hat)
  - International Representative (IR): Barry Hedquist (Perennial).

What is C++11

- C++11 is the ISO C++ standard formally ratified by a 21-0 national vote in August 2011
- C++11 is a major upgrade over C++98/03, with performance and convenience features that make it feel like a new language
- C++0x was the “under-development” name for C++11
- Currently shipping compilers (e.g. GCC C++, Clang C++, IBM C++, and Microsoft C++) already implement most or all C++11 features, and are working on C++14 features

The overall aims of the C++11

- Make C++ a better language for systems programming and library building
  - to build directly on C++'s contributions to programming, rather than providing specialized facilities for a particular sub-community (e.g. numeric computation or Windows-style application development).
- Make C++ easier to teach and learn
  - increased uniformity,
  - stronger guarantees,
  - facilities supportive of novices (there will always be more novices than experts).
- Compatibility constraints
  - very rare existing code breaks

Specific design goals

- Maintain stability and compatibility – don't break old code, and if you absolutely must, don't break it quietly.
- Prefer libraries to language extensions – an ideal not fully successful
- Prefer generality to specialization – focus on improving the abstraction mechanisms (classes, templates, etc.).
- Support both experts and novices – novices can be helped by better libraries and through more general rules; experts need general and efficient features.
- Increase type safety – primarily though facilities that allow programmers to avoid type-unsafe features.
- Improve performance and ability to work directly with hardware – make C++ even better for embedded systems programming and high-performance computation.
- Fit into the real world – consider tool chains, implementation cost, transition problems, Application Binary Interface issues, teaching and learning, etc.

Specific design goals – other perspective

- Machine model and concurrency – provide stronger guarantees for and better facilities for using modern hardware (e.g. multicores and weakly coherent memory models). Examples are threads, futures, thread-local storage, and atomics.
- Generic programming – GP is among the great success stories of C++98; we needed to improve support for it based on experience. Examples are auto and template aliases.
- Systems programming – improve the support for close-to-the-hardware programming (e.g. low-level embedded systems programming) and efficiency. Examples are constexpr, std::array, and generalized PODs.
- Library building – remove limitations, inefficiencies, and irregularities from the abstraction mechanisms. Examples are inline namespace, inherited constructors, and rvalue references.