Welcome to C++ Edinburgh

cppedinburgh.uk

meetup.com/cppedinburgh

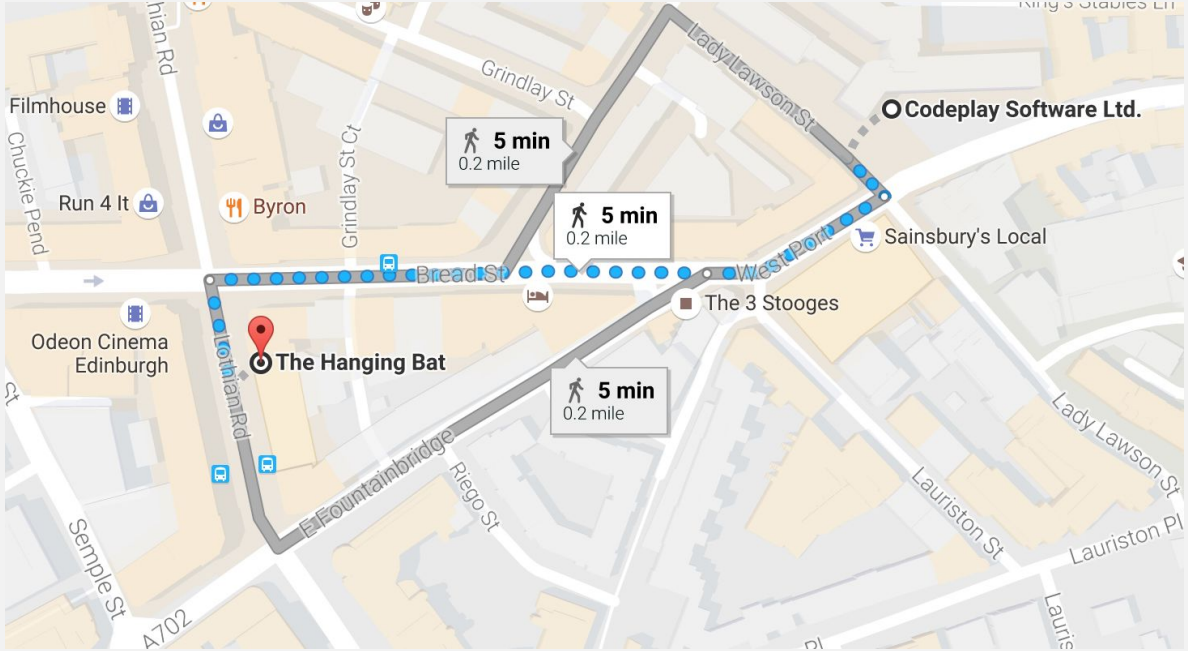
@cppedinburgh

Thanks to our sponsors

Agenda

- Intro and C++ Update
- Composable GPU Programming with Eager Actions and Lazy Views — *Michel Steuwer, University of Edinburgh*
- Why the compiler broke your program — *Peter Brett, LiveCode*
- C++ Binary Dependency Management Using Gradle — *Hugh Greene, Toshiba Medical*
- Drinks at The Hanging Bat

Agenda


Filmhouse
Chuckie Pend
Run 4 It
Byron
Grindlay St Ct
Grindlay St
5 min
0.2 mile
Lady Lawson St
Codeplay Software Ltd.
5 min
0.2 mile
Sainsbury's Local
Bread St
West Port
The 3 Stooges
Odeon Cinema
Edinburgh
The Hanging Bat
Lothian Rd
5 min
0.2 mile
Fountainbridge
Riego St
Lady Lawson St
Lauriston St
Lauriston Pl
Semple St
A702

Keep up-to-date

with C++ Edinburgh happenings.

http://cppedinburgh.uk/

https://meetup.com/cppedinburgh

@cppedinburgh

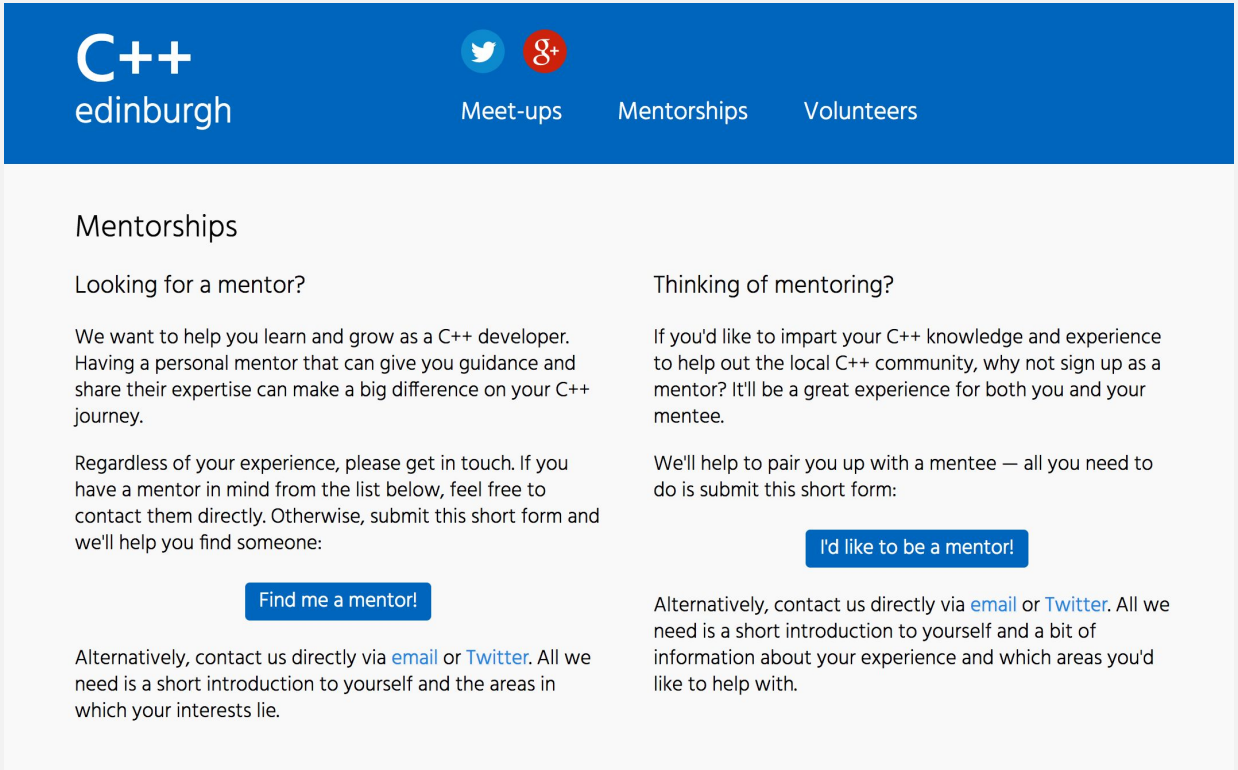
Meet-up schedule

Every second monday of every second month

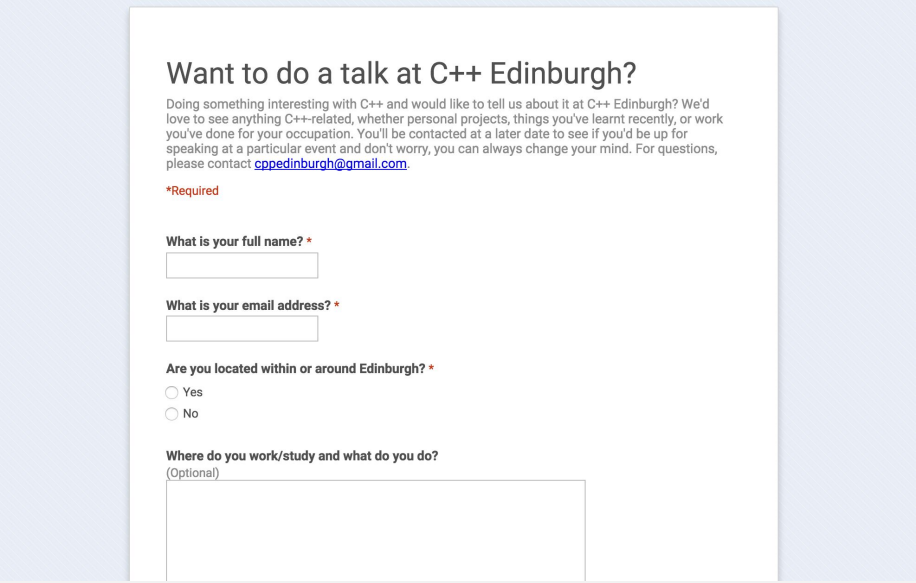
Alternating between normal meet-ups and mini meet-ups

JAN	FEB	MAR	APR	MAY	JUN
JUL	AUG	SEP	OCT	NOV	DEC

Mentorships!



Want to do a talk?



Want to do a talk at C++ Edinburgh?

Doing something interesting with C++ and would like to tell us about it at C++ Edinburgh? We'd love to see anything C++-related, whether personal projects, things you've learnt recently, or work you've done for your occupation. You'll be contacted at a later date to see if you'd be up for speaking at a particular event and don't worry, you can always change your mind. For questions, please contact cppedinburgh@gmail.com.

*Required

What is your full name? *

What is your email address? *

Are you located within or around Edinburgh? *

Yes

No

Where do you work/study and what do you do?

(Optional)

goo.gl/forms/bhS0M2mtGN

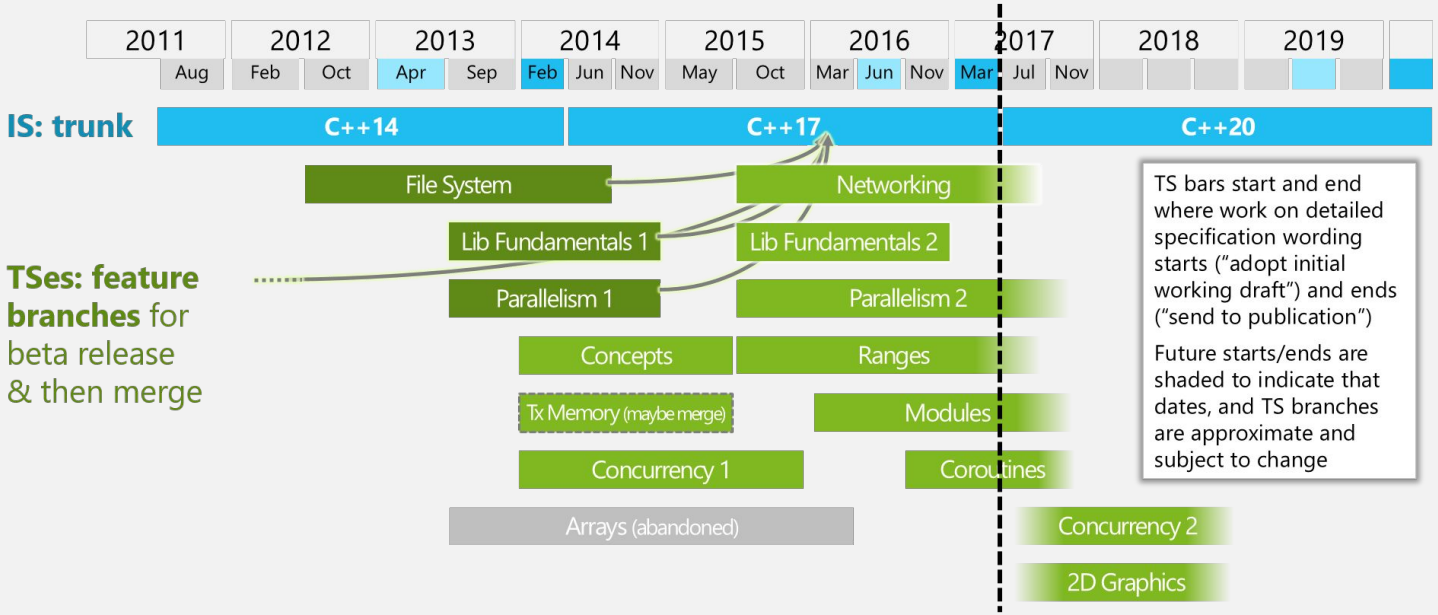
C++ Update
April 2017

Joseph Mansfield
josephmansfield.uk
@sftrabbit

Thanks to our sponsors

Status of C++17

C++17 is now a DIS (Draft International Standard)



Status of C++17

ISO/IEC JTC1 SC22 WG21 P0636r0
Date: 2017-04-02
To: the public
Thomas Köppe <tkoeppe@google.com>

Changes between C++14 and C++17 DIS

Abstract

This document enumerates all the major changes that have been applied to the C++ working draft since the publication of C++14, up to the publication of the C++17 DIS (N4660). Major changes are those that were added in the form of a dedicated paper, excluding those papers that are large issue resolutions. No issue resolutions from either CWG or LWG issues lists ("defect reports") are included.

Contents

Removed or deprecated features

Document	Summary	Examples, notes
N4086	Remove trigraphs	The sequence `??!` no longer means `|`. Implementations may offer trigraph-like features as part of their input encoding.
P0001R1	Remove `register`	The `register` keyword remains reserved, but it no longer has any semantics.
P0002R1	Remove ++ for `bool`	Increment (++) prefix and postfix expressions are no longer valid for operands of type `bool`.
P0003R5	Remove `throw(A, B, C)`	Dynamic exception specifications of the form `throw(A, B, C)` are no longer valid. Only `throw()` remains as a synonym for `noexcept(true)`. Note the change in termination semantics.
P0386R2	Deprecate redeclaration of static constexpr class members	Given `struct X { static constexpr int n = 10; };`, `int X::n;` is no longer a definition, but instead a redundant redeclaration, which is deprecated. The member `X::n` is implicitly inline (see below).
N4190	Remove `auto_ptr`, `random_shuffle`, old parts of `<functional>`	Features that have been deprecated since C++11 and replaced with superior components are no longer included. Their names remain reserved, and implementations may choose to continue to ship the features.
P0004R1	Remove deprecated iostream aliases	Same as above
P0302R1	Remove allocator support from `function`	The polymorphic function wrapper `function` no longer has constructors that accept an allocator. Allocator support for type-erasing, copyable types is difficult, and possibly not implementable efficiently.

New core language features with local applicability

These are features where you would know if you were using them.

Document	Summary	Examples, notes
N4267	A `u8` character literal	A character literal prefix `u8` creates a character that is a valid Unicode code point that takes one code unit of UTF-8, i.e. an ASCII value: `u8'x'`
P0245R1	Hexadecimal floating point literals	Floating point literals with hexadecimal base and decimal exponent: `0xC.68p+2`, `0x1.P-126`. C has supported this syntax since C99, and `printf` supports it via `%a`.
N4295, P0036R0	Fold expressions	A convenient syntax for applying a binary operator iteratively to the elements of a parameter pack: `template <typename ...Args> auto f(Args ...args) { return (0 + ... + args); }`
P0127R2	`template <auto>`	A non-type template parameter may now be declared with placeholder type `auto`. Examples: • `template <auto X> struct constant { static constexpr auto value = X; };` • `Delegate<&MyClass::some_function>`
P0091R3, P0433R2, P0512R0, P0620R0	Class template argument deduction	The template arguments of a class template may now be deduced from a constructor. For example, `pair p(1, 'x');` defines `p` as `pair<int, char>` (this is not an HTML error, the template arguments were omitted deliberately). The implicit deduction is complemented by a system of explicit deduction guides which allow authors to customise how the deduction happens, or forbid it.
P0292R2	Constexpr `if`	In a template specialization, the arms of the new `if constexpr (condition)` statement are only instantiated if the condition (which must be a constant expression) has the appropriate value.
P0305R1	Selection statements with initializer	The selection statements `if` and `switch` gain a new, optional initializer part: `if (auto it = m.find(key); it != m.end()) return it->second;`
P0170R1	Constexpr lambdas	Lambda expressions may now be constant expressions: `auto add = [](int a, int b) constexpr { return a + b; }; int arr[add(1, 2)];`
P0018R3	Lambda capture of `*this`	Before: `[self = *this]{ self.f(); }` Now: `[*this]{ f(); }`
P0386R2	Inline variables	In a header file: `inline int n = 10;` All definitions refer to the same entity. Implied for static constexpr class data members.
P0217R3, P0615R0	Structured bindings	`auto [it, ins] = m.try_emplace(key, a1, a2, a3);` Decomposes arrays, all-members-public classes, and user-defined types that follow a `get<N>` protocol like `pair` and `tuple`.
P0061R1	`__has_include`	A preprocessor operator to check whether an inclusion is possible.
P0188R1	Attribute `[[fallthrough]]`	A new set of standardised attributes. The attributes formally have no required semantics, but implementations are encouraged to emit or suppress the appropriate diagnostics (warnings).
P0189R1	Attribute `[[nodiscard]]`	

https://isocpp.org/files/papers/p0636r0.html

Announcements?
Questions?