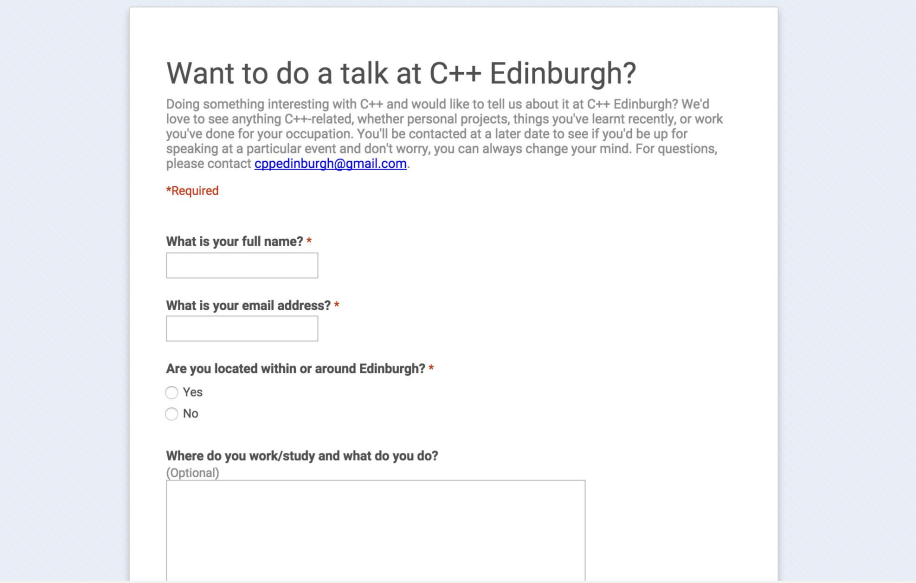
Welcome to C++ Edinburgh

Joseph Mansfield
josephmansfield.uk
@sftrabbit

Thanks to our sponsors

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)

http://goo.gl/forms/bhS0M2mtGN

Keep up-to-date

with C++ Edinburgh happenings.

http://cppedinburgh.uk/

@cppedinburgh

C++ Edinburgh

Mailing List

C++ Update March 2016

Joseph Mansfield
josephmansfield.uk
@sftrabbit

Thanks to our sponsors

Status of C++17

The following things made it in...

Language

- [[fallthrough]], [[nodiscard]], [[maybe_unused]] attributes
- constexpr lambdas
- Generalizing range-based for loops
- Capturing *this in lambdas
- Hexadecimal floating point literals

Library

- (parts of) Library Fundamentals TS v1
- Parallelism TS v1
- File System TS v1
- Special math functions
- hardware_*_interference_size
- .is_always_lockfree()
- clamp()
- non-const .data() for string

Status of C++17

Unfortunately, the following didn't make it

Language

- Uniform call syntax
- Concepts

Library

- Modules
- Coroutines

Status of Technical Specifications

Useful blog post: http://meetingcpp.com/index.php/br/items/c17-and-its-technical-specifications.html

Parallelism TS	Merged into C++17
Filesystem TS	Merged into C++17
Library Fund. TS	Part merged into C++17
Concepts TS	Published
Concurrency TS	Published
Transac. Memory TS	Published
Ranges TS	Working draft
Networking TS	Working draft
Modules TS	New TS - work starting
Coroutines TS	New TS - work starting

Vulkan specification released

Modern, low-level GPU API for graphics and compute

- Nvidia drivers available for Linux/Windows
- AMD drivers available for Windows
- C++ wrapper released by Nvidia: https://github.com/nvpro-pipeline/vkcpp

Browse the C++ standard online

5 Expressions [expr]

5.1 Primary expressions [expr.prim]

5.1.1 General [expr.prim.general]

```
primary-expression:
    literal
    this
    ( expression )
    id-expression
    lambda-expression
    fold-expression

id-expression:
    unqualified-id
    qualified-id

unqualified-id:
    identifier
    operator-function-id
    conversion-function-id
    literal-operator-id
    ~ class-name
    ~ decltype-specifier
    template-id
```

1 A *literal* is a primary expression. Its type depends on its form ([lex.literal]). A string literal is an lvalue; all other literals are prvalues.

2 The keyword `this` names a pointer to the object for which a non-static member function ([class.this]) is invoked or a non-static data member's initializer ([class.mem]) is evaluated.

3 If a declaration declares a member function or member function template of a class `X`, the expression `this` is a prvalue of type "pointer to *cv-qualifier-seq* X" between the optional *cv-qualifer-seq* and the end of the *function-definition*, *member-declarator*, or *declarator*. It shall not appear before the optional *cv-qualifier-seq* and it shall not appear within the declaration of a static member function (although its type and value category are defined within a static member function as they are within a non-static member function). [*Note:* this is because declaration matching does not occur until the complete declarator is known. — *end note*] Unlike the object expression in other contexts, `*this` is not required to be of complete type for purposes of class member access ([expr.ref]) outside the member function body. [*Note:* only class members declared prior to the declaration are visible. — *end note*] [*Example:*

```
struct A {
  char g();
  template<class T> auto f(T t) -> decltype(t + g())
    { return t + g(); }
};
```

http://eel.is/c++draft/

Announcements?
Questions?