Language Features of C++17

New auto rules for direct-list-initialization

`auto x { 1 };` will be now deduced as `int`, but before it was an initializer list. For a braced-init-list with only a single element, auto deduction will deduce from that entry; For a braced-init-list with more than one element, auto deduction will be ill-formed.

Typename in a template template parameter

You can now use `typename` instead of `class` when declaring a template template parameter.

Nested namespace definition

Allows you to write:

```
namespace A::B::C { /* … */ }
```

Rather than:

```
namespace A { namespace B { namespace C {/* … */ }}}
```

Fold Expressions

Allows you to write compact code with variadic templates without using explicit recursion.

```
template<typename... Args>
auto SumAll(Args... args){ return (args + ...); }
```

Unary fold expressions and empty param packs

Specifies what to do when the parameter pack is empty for operators: &&, || and comma. For other operators we get invalid syntax.

Removing Deprecated Exception Specifications

Dynamic exception specifications were deprecated in C++11. In C++17 the feature is removed while retaining the (still) deprecated `throw()` specification strictly as an alias for `noexcept(true)`.

Exception specifications part of the type system

Previously exception specifications for a function didn't belong to the type of the function, but it will be part of it.

Aggregate initialization of classes with base classes

If a class was derived from some other type you couldn't use aggregate initialization. But now the restriction is removed.

Lambda capture of *this

this pointer is implicitly captured by lambdas inside member functions. Now you can use ***this** when declaring a lambda and this will create a copy of the object. Capturing by value might be especially important for async invocation, parallel processing.

Memory allocation for over-aligned data

C++11/14 did not specify any mechanism by which over-aligned data can be dynamically allocated (i.e. respecting the alignment of the data). Now, we get new functions that takes alignment parameters. Like `void* operator new(std::size_t, std::align_val_t);`

__has_include in preprocessor conditionals

This feature allows a C++ program to directly, reliably and portably determine whether or not a library header is available for inclusion.

Template argument deduction for class templates

Before C++17, template deduction worked for functions but not for classes. `std::pair intChar{42, ‘c’};` is now deduced as `std::pair<int, char>` in C++17.

Non-type template parameters with auto type

Automatically deduce type on non-type template parameters.

```
template <auto value> void f() { }
f<10>(); // deduces int
```

Guaranteed copy elision

Copy elision (e.g. RVO) was a common compiler optimization, now it's guaranteed and defined by the standard!

Direct-list-initialization of enumerations

You can now initialize `enum class` with a fixed underlying type:

```
enum class Handle : uint32_t { Invalid = 0 };
Handle h { 42 }; // OK
```

Stricter expression evaluation order

In expression such as `f(a, b, c)`: the order of evaluation of `a`, `b`, `c` is still unspecified, but any parameter is fully evaluated before the next one is started. Plus other "practical" changes:

⇨ Postfix expressions are evaluated from left to right.

⇨ Assignment expressions are evaluated from right to left.

⇨ Operands to shift operators are evaluated from left to right.

The code below now evaluates as `f`, `h`, `g`, `I` (previously any order)

```
std::cout << f() << g(h()) << i();
```

constexpr lambda expressions

constexpr can be used in the context of lambdas.

```
constexpr auto ID = [] (int n) { return n; };
static_assert(ID(3) == 3);
```

Differing begin and end types in range-based for

Types of `__begin` and `__end` iterators (used in the loop) will be different; only the comparison operator is required. This little change improves Range TS experience.

Pack expansions in using-declarations

Allows you to inject names with using-declarations from all types in a parameter pack.

```
template<class... Ts> struct overloaded : Ts... {
using Ts::operator()...; };
```

constexpr if-statements

The static-if for C++! Reduces the need to use SFINAE or tag dispatch.

```
if constexpr (is_floating_point_v<T>) { }
```

Attribute Features

[[fallthrough]] - indicates that a case in a switch statement can fall-through.

[[nodiscard]] - specifies that a return value should not be discarded, there's warning reported otherwise.

[[maybe_unused]] - the compiler will not warn about a variable that is not used.

Ignore unknown attributes - compilers which don't support a given attribute will ignore it. Previously it was unspecified.

Using attribute namespaces without repetition – simplifies using attributes from the same namespace

Attributes for namespaces and enumerators – Fixes the spec, so now attributes can be used for most of the declarations, variables, classes, enums, namespaces, enum values, etc.

Structured Bindings

Automatically decomposes packed structures like tuples structs and arrays into individual named variables.

```
auto [ a, b, c ] = tuple; // or struct or array
```

Init-statements for if and switch

```
if (auto val = GetValue(); condition(val))
    // on success
else
    // on false...
```

`val` is only present in the scope of the `if` and the `else` clause.

Inline variables

Variables can be declared inline in the same way as inline functions.

```
class MyClass {
   static inline const std::string s_val = "Hello";
```

Other

⇨ **static_assert with no message**

⇨ **u8 character literals**

⇨ **Removing trigraphs**

⇨ **Remove Deprecated Use of the register Keyword**

⇨ **Remove Deprecated operator++(bool)**

⇨ **Hexadecimal floating-point literals**

⇨ **Allow constant evaluation for all non-type template arguments**

⇨ **New specification for inheriting constructors**

⇨ **Matching of template template-arguments update**

⇨ **Removal of std::auto_ptr, std::random_shuffle, and more**

References

http://www.bfilipek.com/2017/01/cpp17features.html, https://isocpp.org/, https://herbsutter.com/, http://en.cppreference.com/w/cpp/compiler_support, http://baptiste-wicht.com/, https://tartanllama.github.io/, https://jonasdevlieghere.com/,

https://leanpub.com/cpp17indetail

Last update July 2019