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

Allows you to use typename instead of class when declaring a template template parameter.

Nested namespace definition

Allows to write:

```
namespace A::B::C { /* … */ }
```

Rather than:

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

Fold Expressions

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

```
template<typename... Args>
auto SumWithOne(Args... args){
    return (1 + ... + 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. This paper formally proposes removing the feature from C++17, 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. 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 correctly (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. For `void f(std::pair<int, char>);` you had to explicitly write `f(std::pair<int, char>(42, 'z'));` Now the restriction is removed and `f(std::pair(42, 'z'));` will work.

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

Allows to initialize enum class with a fixed underlying type:

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

Stricter expression evaluation order

An expression such as f(a, b, c), the order in which the sub-expressions f, a, b, c (which are of arbitrary shapes) are evaluated is left unspecified by the standard.

Summary of 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.

This should fix problem with function chaining, std::then(), stream <<

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.

std::uncaught_exceptions()

The function returns the number of uncaught exception objects in the current thread. This might be useful when implementing proper Scope Guards that works also during stack unwinding.

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 in most cases.

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 if and else clause.

Inline variables

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

constexpr if-statements

The static-if for C++! Will reduce need to use complicated SFINAE or tag dispatch.

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/,