C++20 Features

New keywords: `char8_t`, `co_await`, `co_return`, `co_yield`, `concept`, `consteval`, `constinit`, `import*`, `module*`, `requires`
* identifiers with a special meaning

Concepts

Constrains on the template parameters and meaningful compiler messages in case on an error. Can also reduce the compilation time.

```
template <class T>
concept SignedIntegral = std::is_integral_v<T> &&
                         std::is_signed_v<T>;
template <SignedIntegral T> // no SFINAE here!
void signedIntsOnly(T val) { }
```

Modules

The replacement of the header files! With modules you can divide your program into logical parts.

```
import helloworld; // contains the hello() function
int main() {
   hello(); // imported from the "helloworld" module!
}
```

Coroutines

Functions that can suspend their execution and can be resumed later, also asynchronously. They are associated with a promise object and might be allocated on the heap. C++20 gives language support. Use libs like cppcoro for full functionality (generators objects).

```
generator<int> iota(int n = 0) {
  while(true)
    co_yield n++;
}
```

operator<=>

New operator that can define other operators: <, <=, >, and >=.

`R operator<=>(T,T);` where R can be: `std::strong_ordering`, `std::weak_ordering` and `std::partial_ordering`.

```
 (a <=> b) < 0 if a < b
(a <=> b) > 0 if a > b
(a <=> b) == 0 if a and b are equal/equivalent.
```

Designated Initializers

Explicit member names in the initializer expression:

```
struct S { int a; int b; int c; };
S test {.a = 1, .b = 10, .c = 2};
```

Range-based for with initializer

Create another variable in the scope of the for loop:

```
for (int i = 0; const auto& x : get_collection()) {
    doSomething(x, i);
    ++i;
}
```

char8_t

Separate type for UTF-8 character representation, the underlying type is `unsigned char`, but they are both distinct. The Library also defines now `std::u8string`.

Attributes

`[[likely]]` - guides the compiler about more likely code path
`[[unlikely]]` - guides the compiler about uncommon code path
`[[no_unique_address]]` - useful for optimisations, like EBO
`[[nodiscard]]` for constructors – allows us to declare the constructor with the attribute. Useful for ctors with side effects, or RAII.
`[[nodiscard("with message")]]` – provide extra info
`[[nodiscard]]` is also applied in many places in the Standard Library

Structured Bindings Updates

Structured bindings since C++20 are more like regular variables, you can apply `static`, `thread_storage` or capture in a lambda.

Class non-type template parameters

Before C++20 only integral types, enums, pointer and reference types could be used in non-type template parameters. In C++20 it's extended to classes that are Literal Types and have "*structural equality*".

```
struct S { int i; };
template <S par> int foo() { return par.i + 10; }
auto result = foo<S{42}>();
```

explicit(bool)

Cleaner way to express if a constructor or a conversion function should be `explicit`. Useful for wrapper classes. Reduces the code duplication and SFINAE.

```
explicit(!is_convertible_v<T, int>) ...
```

constexpr Updates

`constexpr` is more relaxed you can use it for `union`, `try` and `catch`, `dynamic_cast`, memory allocations, `typeid`. The update allows us to create `constexpr std::vector` and `std::string` (also part of C++ Standard Library changes)! There are also `constexpr` algorithms like `std::sort`, `std::rotate`, `std::reverse` and many more.

consteval

A new keyword that specifies an immediate function – functions that produce constant values, at compile time only. In contrast to `constexpr` functions, they cannot be called at runtime.

```
consteval int add(int a, int b) { return a+b; }
constexpr int r = add(100, 300);
```

constinit

Applied on variables with static or thread storage duration, ensures that the variable is initialized at compile-time. Solves the problem of static order initialisation fiasco for non-dynamic initialisation. Later the value of the variable can change.

Ranges

A radical change how we work with collections! Rather than use two iterators, we can work with a sequence represented by a single object.

```
std::vector v { 2, 8, 4, 1, 9, 3, 7, 5, 4 };
std::ranges::sort(v);
for (auto& i: v | ranges:view::reverse) cout << i;
```

With Ranges we also get new algorithms, views and adapters

std::format

Python like formatting library in the Standard Library!

```
auto s = std::format("{:-^5}, {:-<5}", 7, 9);
```

`s` has a value of „`--7--, 9----`" centred, and then left aligned
Also supports the Chrono library and can print dates

Chrono Calendar, Timezone and Updates

Heavily updated with Calendar and Timezones

```
auto now = system_clock::now();
auto cy = year_month_day{floor<days>(now)}.year();
cout << "The current year is " << cy << '\n';
```

Additionally we have updates like explicit `file_clock`, `clock_cast` (time point conversion) and many other enhancements.

Multithreading and Concurrency

- `jthread` - automatically joins on destruction. Stop tokens allows more control over the thread execution.
- More atomics: floats, `shared_ptr`, `weak_ptr`, `atomic_ref`
- Latches, semaphores and barriers – new synchronisation primitives

std::span

A on owning contiguous sequence of elements. Unlike `string_view`, `span` is mutable and can change the elements that it points to.

```
vector<int> vec = {1, 2, 3, 4};
span<int> spanVec (vec);
for(auto && v : spanVec) v *= v;
```

Other

- Class Template Argument Deduction for aliases and aggregates, and more CTAD in the Standard Library
- template-parameter-list for generic lambdas
- Make `typename` optional in more places
- Signed integers are two's complement
- `using enum` – less typing for long `enum class` names
- Deprecating `volatile` where it has no obvious meaning.
- Pack expansion in lambda init-capture
- `std::bind_front()` - replacement for `std::bind()`
- String prefix and suffix checking
- `std::bit_cast()` and bit operations
- Heterogeneous lookup for unordered containers
- `std::lerp()` and `std::midpoint()`, Math constants
- `std::source_location()` – get file/line pos without macros
- Efficient sized delete for variable sized classes
- Feature test macros and the `<version>` header
- `erase/erase_if` non-member functions for most of containers!

References

isocpp.org, herbsutter.com,
en.cppreference.com/w/cpp/compiler_support,
devblogs.microsoft.com/cppblog/c20-concepts-are-here...,
C++20: the small things - Timur Doumler - Meeting C++ 2019