Hi,

Welcome to a summary of news, articles, videos and resources from December 2019 that might be valuable to you.

This month I'd like to draw your attention to the following topics:

- C++ at the end of the Year
- Short Circuit in Meta Functions
- ACCU Overload Magazine, December Edition
- Chrono Library Design and Improvements for C++20
- Visual Studio 16.4 Update
- and more...

Let's start!

News and Updates

This section contains short news and updates that happened in December:

- **C++ at the end of 2019** - read the full article that summarises the whole Year for C++. What happened, how the C++ standard progressed towards C++20, what are the new tools, books and much more.

  Here's the link: C++ at the end of 2019 @bfilipek.com

- Meeting C++ 2019 conference videos are available - you can now watch all the videos that were recorded at the last Meeting C++ conference. Lot's of good material to see!

  Link to the YouTube Channel

- Avast December C++ Meetup with Chandler Carruth. See the YouTube recording here:

  Chandler Carruth - Programming Language Design for Performance Critical Software

- Qt 5.14 is ready! Also with Qt Creator 4.11. See the blog post on that: Qt 5.14 Released!

- Visual Studio 2019 16.4 released. Including clang-tidy integration, vertical tabs, ASAN, Build Insights and many more! See the release notes on their blog. 'Tis the Season for the Visual Studio 2019 v16.4 Release

Articles, Videos & Resources

In this section, we'll have a closer look at some recent content about C++.

Short Circuit in Meta functions

Short-circuiting in meta-functions by Andrzej Krzemienski

In the article Andrzej describes the difference between evaluation of compound expressions in conditions like:

```
if (ptr && ptr->is_ready())
```

and

```
if constexpr(std::is_trivial<C>::value &&
             std::is_trivial<D>::value &&
             has_alternate<C>::value  &&
             has_alternate<D>::value)
```

In the first example, we can assume that if `ptr` is null, then the following expression won't be evaluated. But is the second example, even if the first template expression is false, then still all other expressions are evaluated at compile time.

The author then presents `std::conjunction` that allows the use of short circuit behaviour. That way we can save some compilation time for larger expressions.

In the end, you'll learn that in C++20 we'll have `requires` keyword that uses short circuit and provides lazy evaluation.

About Chrono Design and C++20 Improvements

In December I watched an excellent presentation from Howard Hinnat about the **Chrono library**. This was also the keynote on Metting C++ 2019.

As you may know, in C++20, this library will be heavily extended - with formatting, parsing and calendar support! Howard described all the principles behind the library, its goals, flexibility and the C++20 overview. We can now say that Chrono is complete and the plan is that you won't ever need to touch anything outside this library to work with time and dates and time!

Here's the video: Opening Keynote Meeting C++ 2019 - Howard Hinnant - Design Rationale for the Chrono Library

Overload 154 is now available

ACCU's Overload journal of December 2019 is out. It contains the following articles:

- Inside-Out
- Trip Reports: Meeting C++ 2019 and Embedded C++ 2019
- Non-Recursive Compile Time Sort
- Quick Modular Calculations (Part 1)

The magazine is free, and you can download it as a single pdf or read it online.

ACCU :: Overload 154

CMake - Unity Build & Precompiled Headers

CMake 3.16 added support for precompiled headers & unity builds - what you need to know - Not just gamedev

by Viktor Kirilov

Unity build is a popular technique to save compilation time. The primary approach is to have a single (or several) cpp file that includes other cpp files. This way the compiler has only a single or a few compilation units and thus it has much less work to do.

On the other hand, the precompiled headers technique is common in Visual Studio. It precompiles all the "common" header files and stores the intermediate results in some cache. Later, when you build the project, the compiler doesn't need to process the headers and just uses the cache. It's useful to store common library headers, STL headers and headers that are already "solid" and don't change.

In the article, you can learn how to use those techniques through the latest release of CMake.

C++ Weekly - [[nodiscard]] constructor

C++ Weekly - Ep 199 - nodiscard - Constructors And Their Uses - YouTube

by Jason Turner

This is a short video that demonstrates the difference between applying `[[nodiscard]]` on the whole type and applying it on the constructor.

```
struct [[nodiscard]] S { }; // C++17
```

Since C++17 you can mark the whole type as `[[nodiscard]]` and then if an expression returns a value of that type the compiler will generate a warning.

But in C++17, it was not possible to mark individual constructors with this attribute. It's fixed in C++20, and now you can write:

```
struct S {
    [[nodiscard]] S(int) { } // since C++20!
}
```

Now, the compiler will warn if you write `S{10};`. The main difference here is that applying `[[nodiscard]]` on constructors might be necessary to types that do something in the constructor: especially acquiring some resource, but the whole type doesn't have to be `[[nodiscard]]`. For example `unique_ptr` or locks.

Avoid `std::bind`

abseil / Tip of the Week #108: Avoid `std::bind`

on the Google Abseil Blog

`std::bind` available from C++11 replaced `std::bind1st` and `std::bind2nd`. Those helpers are handy for partial function application.

However, the blog post explains that `std::bind` is still not perfect as you need to specify the full number of arguments, for example, by using the placeholder notation. There might also be issues with the number of arguments passed, in such cases, you might not get compiler errors, and the code will silently do the wrong things. Plus it's hard to use when you want to pass moveable only types like `unique_ptr`.

Since C++20 we'll have a new tool: `std::bind_front` that avoids many of the issues of `std::bind` and is simpler to use. It binds the first N arguments and perfect-forwards the rest In the Abseil library you can even use that before C++20 is accepted, just use `absl::bind_front`.

Extra: How Do Bullets Work in Video Games

Gamasutra: Tristan Jung's Blog - How Do Bullets Work in Video Games?

Something not directly related to C++ :)

This is a very well written article that explains how games do bullets. It looks like there are two approaches to that problem: hitscan and Projectile Ballistics.

The first one assumes the bullet has no mass, and it travels at "infinite" speed. It's simple but efficient and handy for machine guns or laser shots.

The second approach involves real physics: each time a player shoots the engine creates a real projectile and uses physical simulations to do the job.

Both approaches have pros and cons, and in modern engines, you can even select between them or use a hybrid technique.

About the Author

Bartlomiej Filipek - @fenbf - is a C++ software developer with more than 12 years of professional experience, Author ("C++17 in Detail"), MVP and C++ ISO Member. In 2010 I graduated from Jagiellonian University in Cracow with a Masters Degree in Computer Science.

Bartek works at Xara, where he develops features for advanced document editors. Bartek has also experience with desktop graphics applications, game development, large-scale systems for aviation, writing graphics drivers and even biofeedback. In the past, he also taught programming (mostly game and graphics programming courses) at local universities in Cracow.

Since 2011 Bartek has been regularly blogging at bfilipek.com. In the early days, the topic revolved around graphics programming, and now he focuses on Core C++. Bartek is also a co-organizer of the C++ User Group in Cracow.