# C++ Club UK Meeting 145

Gleb Dolgich

2022-03-17

## Contents

### War

Discussing C++ while a war in Europe is raging feels surreal. I'm with the people of Ukraine who are experiencing this unspeakable tragedy. I hope they prevail, and I hope the Russian war criminals will be brought to justice. My parents are in Lithuania, and I'm very worried.

### P2300 is headed to C++26

P2300 `std::execution` is headed to C++26. The latest poll with the question "Advance P2300R5 to electronic polling to send it to LWG for C++26" resulted in strong consensus with just a single neutral vote and no votes against. You probably remember that on its way to C++23 the paper was met with strong objections from quite a few people. Maybe the latest poll reflects the fact that C++26 is a loo-o-o-ng way away and the paper can be ready in time. Or maybe not all committee members were notified of the poll, which I guess is one way of solving the problem.

### Modern C++ Course from Bonn U

There is a decent Modern C++ course from Bonn University on YouTube, check it out.

### Mold 1.1.1 released

Rui Ueyama released version 1.1.1 of his new fast linker mold. This version adds new LTO options for lld compatibility and reduces memory usage by ~6%, consuming less memory than GNU Gold or Clang lld.

## String-like parameter cheatsheet

**hacking C++** @hackingcpp
👥 @fenbf

also for regular functions my rules are:

### String(-Like) Function Parameters

| If You... | Use Parameter Type |
|---|---|
| *always* need a copy of the input string inside the function | `std::string` "pass by value" |
| want **read-only access** • don't (always) need a copy • are using **C++17/20** | #include <string_view> `std::string_view` |
| want **read-only access** • don't (always) need a copy • are stuck with **C++98/11/14** | `std::string const&` "pass by const reference" |
| want the function to **modify the input string in-place** (you should try to avoid such "output parameters") | `std::string &` "pass by (non-const) reference" |

💬 thread
1d • 12/12/2021 • 19:18   ♥

When using the pass-by-value + move idiom in constructors, be extra careful not to use the passed parameter after it has been moved-from.

## Specializations of variable templates can have different type

Eric Niebler tweets:

> Specializations of a variable template can have different types. Huh. #TIL #CPP

Hana Dusíková replies:

> It's exactly [the] same as specialization of [a] template based on type

🇺🇦 **Eric Niebler** 🇺🇦 **#BLM** @ericniebler
10/03/2022, 00:03 Twitter Web App

Specializations of a variable template can have different types. Huh. #TIL #CPP

```cpp
template <auto X>
inline constexpr auto foo = X;

template <>
inline constexpr char const* foo<42> = "the answer";

bool b = foo<true>;
char const* c = foo<42>;
```

💬 9        ↻ 9        ❤️ 60        🔖        ⬆️        •••

🇺🇦 **Hana Dusíková** 👩🏻 🇺🇦
@hankadusikova

10/03/2022, 13:28 Twitter Web App
Replying to @ericniebler

It's exactly same as specialization of template based on type

template <typename T> struct hana_type { };
template <> struct hana_type<std::string>: std::string { };

#yolo

💬        ↻        ♡ 4        🔖        ⬆️        •••

Corentin Jabot follows up:

4

One of the things that's currently bending my mind is that you can have a template variable which is a generic lambda

```
1  template <class>
2  auto x = []<class>{};
```

## Secure coding practices

Amir Kirsh posted an article on the IncrediBuild blog called *Top 10 secure C++ coding practices*. In it he gives an overview of what security is and how a C++ programmer can make their code more robust to avoid vulnerabilities. He starts with the following:

Understand that there are no safety nets provided by the compiler or runtime while coding in C++. C++ compiler generates the code the programmer asked it to generate, without adding any safety checks. While coding in C# or Java, for example, incorrect array access would lead to a runtime exception, whereas in C++ this leads to incorrect memory access or memory corruption in case of writing. Incorrect or sloppy coding can lead to overflows (stack, heap, and buffer overflows) which can easily be used for an attack.

Some of the advice from the author:

- Don't misuse APIs. Don't rely on undocumented behaviour. Don't use APIs that are established to be vulnerable.
- Validate input.
- Take advantage of type safety. Don't intentionally bypass type checking.
- Be careful of arithmetic overflows and underflows. (*Ah yes, the infamous* `size_t`)
- Handle exceptions and errors carefully.
  - Don't leak sensitive information including error codes, stack traces, user IDs etc.
- Initialize variables.
- Security by obscurity is no security.
- Don't implement your own cryptography.
- Be careful with random numbers. Use the new C++11 random generators (*but not like that – see P0205*).
  - Don't use uninitialized variables as a random number generator (*What?*)
- Use C++ secure coding standard to complement your C++ coding standard, like SEI Cert C++.
- Use the right tools to detect security issues: static code analysers, sanitizers.

The related Reddit thread has an interesting discussion on using `at()` vs. `[]`. I didn't know that in some cases the compiler can optimize away bounds checks

in `at()`. Of course, a better solution is to use range-`for` loops or even better, ranges and algorithms.

## An interesting clang-tidy bug

Lesley Lai tweeted:
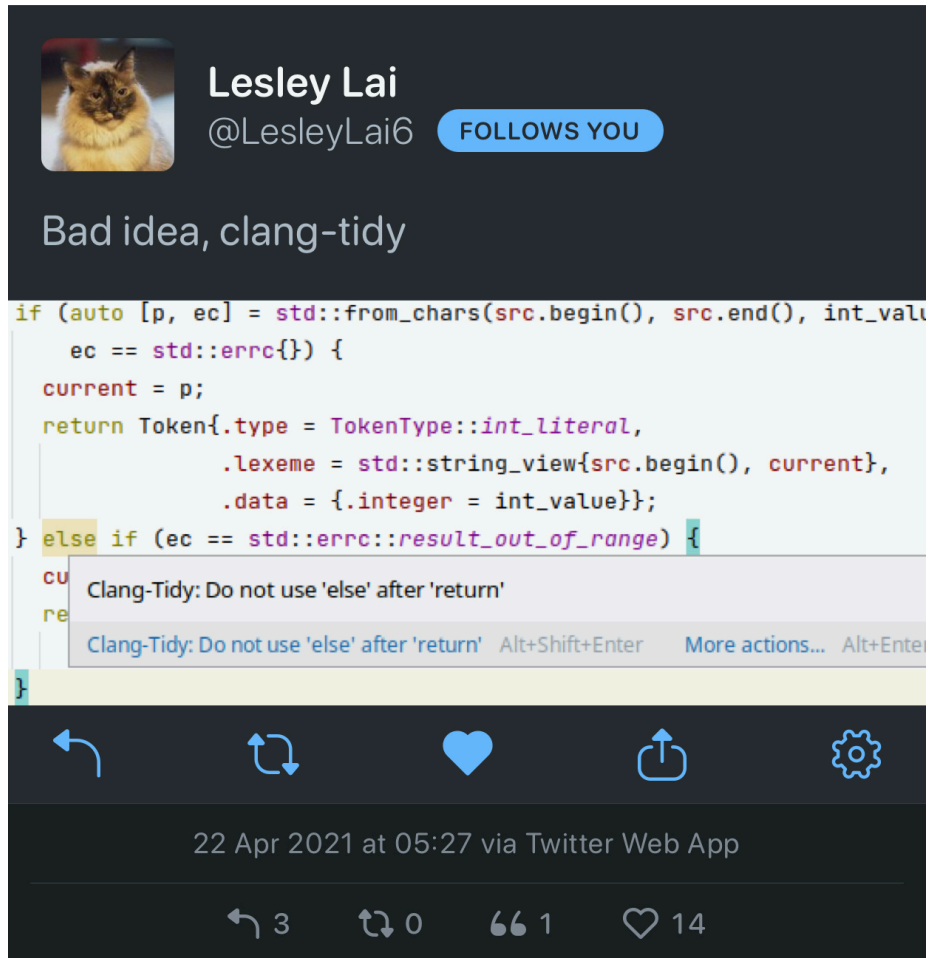


If we applied the fix, the second `if` wouldn't compile because `ec` was declared in the first `if` init statement.

## Twitter

Vicki Boykis (@vboykis) tweets:

**Vicki**
@vboykis

25/02/2022, 16:59 via Twitter Web App

A senior developer is someone who fluently hates more than one programming language.

💬 243     🔁 637     ❤️ 4,861     🔖          ⬆️          ···

TerraTech (@gaya_tech) on embedded programming:

**TerraTech**
@gaya_tech

Microcontroller programming: If the timer overflows, we restart the chip to prevent erratic behavior

FPGA programming: This 84 bit timer should last till the sun explodes

↩️          🔁          ❤️          ⬆️          ⚙️

5 Feb 2022 at 01:41 via Twitter Web App

↩️ 6     🔁 95     ❝❝ 2     ♡ 838