C++ Club UK

Gleb Dolgich

2019-05-23

PyTorch and C++

- Peter Goldsborough (Facebook Research)
  - CppCon 2017: A Tour of Deep Learning With C++
  - CppCon 2018: Machine Learning in C++ with PyTorch
  - Code https://github.com/goldsborough?tab=repositories
- PyTorch docs https://pytorch.org/cppdocs
- PyTorch code https://github.com/pytorch/pytorch

Quirks in Class Template Argument Deduction (1/2)

Barry Revzin: https://brevzin.github.io/c++/2018/09/01/quirks-ctad/

```
1 std::tuple<int> foo();
2
3 std::tuple x = foo(); // tuple<tuple<int>>?
4 auto y = foo();       // tuple<int>
```

What is the intent behind the declaration of variable `x`? Are we constructing a new thing (the CTAD goal) or are we using `std::tuple` as annotation to ensure that `x` is in fact a `tuple` (the Concepts goal)?

Quirks in Class Template Argument Deduction (2/2)

A clearer example:

```
1 // The tuple case
2 // unquestionably, tuple<int>
3 std::tuple a(1);
4
5 // unquestionably, tuple<tuple<int>,tuple<int>>
6 std::tuple b(a, a);
7
8 // ??
9 std::tuple c(a);
```

clamp_cast -- A saturating arithmetic cast

https://github.com/p-groarke/clamp_cast

A narrowing cast that does the right thing. clamp_cast will saturate output values at min or max if the input value would overflow / underflow.

```
1 double ld = -42.0;
2 unsigned char uc = clamp_cast<unsigned char>(ld);
3 // uc == 0
4
5 float f = 500000.f;
6 char c = clamp_cast<char>(f);
7 // c == 127
```

A pretty big list of C++ GUI libraries

Philippe M. Groarke: https://philippegroarke.com/posts/2018/c++_ui_solutions/

Reddit:

- https://www.reddit.com/r/cpp/comments/babfl5/a_pretty_big_list_of_c_gui_libraries/
- https://www.reddit.com/r/cpp/comments/9njw5n/is_there_an_easytouse_gui_library/
- https://www.reddit.com/r/cpp/comments/9q07bu/any_library_as_small_as_wxwidgets_but_as_powerful/

Modern UI in C++
https://www.reddit.com/r/cpp/comments/b3s2zq/modern_ui_in_c/

C++ Experts, what advice would you give to a new C++ developer?

https://www.reddit.com/r/cpp/comments/9s34p9/c_experts_what_advice_would_you_give_to_a_new_c/

- Exceptions don't work with MPI #
- Prefer composition over inheritance
- "Don't overuse exceptions" may not be so clear cut # [Measuring execution performance of C++ exceptions vs error codes]
- Write unit tests for public API # #
- Not everything needs to be a class #
- Consider data-oriented design #
- A bunch of useful tips #
- "Rust is a good choice!" # # *(there's always one or two)*

What are some things commonly taught in C++ that are really bad practice?

https://www.reddit.com/r/cpp/comments/bgdawr/what_are_some_things_commonly_taught_in_c_that/

- Using inheritance for code reuse. After a couple of years you have an unmaintainable spaghetti that goes 5 levels deep. #
- Raw pointers/new/delete without RAII, improper use of raw (C) strings and arrays #
- Trust the programmer. I trusted myself once, and it didn't end well. Never again making that mistake. #
- `using namespace std;` #
- Abuse of `protected`. Where author of base class assumes you will correctly fiddle with protected members. #
- Single entry, single exit. #
- Throwing exceptions (!) #

Same function parameters with different return type in C++17/C++20 (1/3)

https://www.reddit.com/r/cpp/comments/aoidsi/what_is_the_solution_for_same_function_parameters/

Before:

```
1 template<typename R>
2 R foo(int i)
3 { ... }
4
5 foo<string>(1);
```

Same function parameters with different return type in C++17/C++20 (2/3)

https://www.reddit.com/r/cpp/comments/aoidsi/what_is_the_solution_for_same_function_parameters/

After:

```
template<class F> struct Auto : F {
    // conversion operator
    template<class T> operator T() {
        return F::template operator()<T>();
    }
};

template<class F> Auto(F) -> Auto<F>; // deduction guide
```

Same function parameters with different return type in C++17/C++20 (3/3)

https://www.reddit.com/r/cpp/comments/aoidsi/what_is_the_solution_for_same_function_parameters/

After:

```
template<class... A>
auto fooWrapper(A&&... a) {
    return Auto{[&]<class T>() { return foo<T>(std::forward<A>(a)...); }};
};

template<class... A>
auto fooWrapper(int i) {
    return Auto{[=]<class T>() { return foo<T>(i); }};
};

double d = fooWrapper(42);
```

Data alignment the C++ way

https://vorbrodt.blog/2019/04/06/data-alignment-the-c-way/

Before modern C++:

```
1 struct Old
2 {
3     int x;
4     char padding[16 - sizeof(int)];
5 };
```

Now:

```
1 struct alignas(16) New
2 {
3     int x;
4 };
```

Modern Enums

https://www.reddit.com/r/cpp/comments/b9xb3n/its_2019_we_have_the_power_of_constexpr_and/

- Static Enum https://github.com/KonanM/static_enum
- Magic Enum: Enum-to-String and String-to-Enum functions for modern C++ https://github.com/Neargye/magic_enum
- Better Enums http://aantron.github.io/better-enums/
- Wise Enum https://github.com/quicknir/wise_enum
- Meta Enum https://github.com/therocode/meta_enum

Nameof operator for modern C++

https://github.com/Neargye/nameof

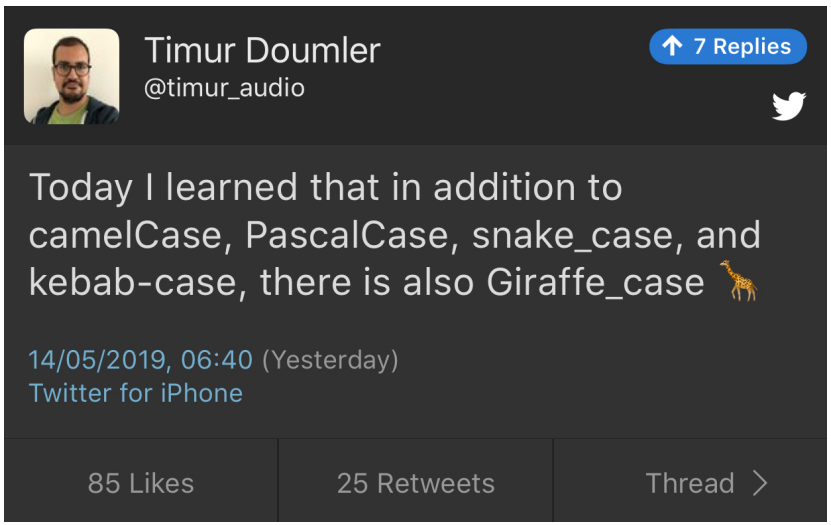
See also: CTTI https://github.com/Manu343726/ctti

Is Microsoft/GSL still being maintained?

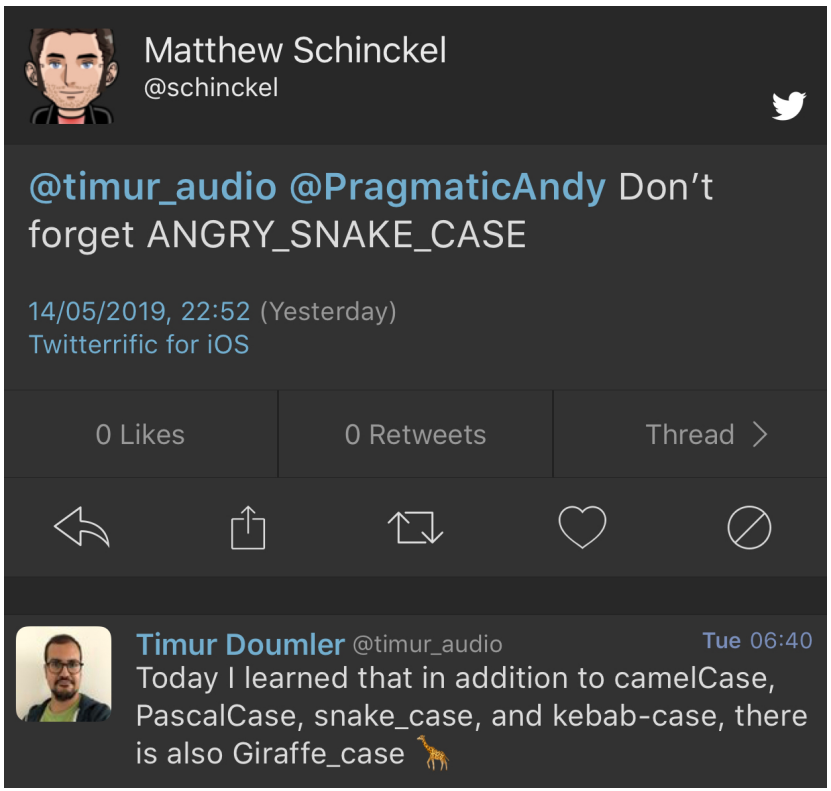
It is used by the brand new Terminal App. That alone is an indication of effort.

- Code: https://github.com/microsoft/GSL
- Reddit: https://www.reddit.com/r/cpp/comments/bmmplo/is_microsoftgsl_still_being_maintained/

Twitter: identifier case



Twitter: identifier case



Twitter: identifier case