C++ Club

27 July 2017

Presenting effectively in meetings

Dirk Haun, ACCU 2017

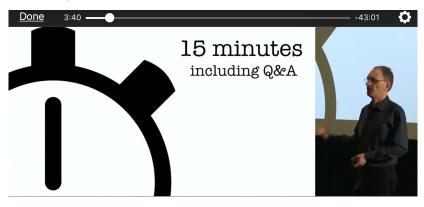


Figure 1: 80%

Roger Orr - Making Templates Easier (ACCU)

Meetup | Video

- Templates
- ▶ enable_if
- Concepts

CLion 2017.3 roadmap

Post

- Valgrind integration
- Remote development support
- Better C++ language support
- Multiple toolchains
- Custom pretty printers for debugger
- GDB 8, LLDB 5
- Gutter icons for unit tests
- Improvements and fixes for Google Test and Catch

Thoughts on Metaclasses - Herb Sutter - Keynote [ACCU 2017]

YouTube | Post

- Depends on:
 - ▶ C++17
 - concepts (C++20)
 - constexpr, if constexpr
 - compile-time meta-programming (P0589, P0633)
 - reflection (P0194, P0385, P0578, P0590, P0598)
- Start watching at 18m

Metaclasses (cont.)

Reflection

```
1 | $T, $expr
```

Compile-time programming

YouTube



void_t and nonesuch

```
template <class...>
    using void_t = void;

struct nonesuch {
    nonesuch() = delete;
    ~nonesuch() = delete;
    nonesuch(nonesuch const&) = delete;
    void operator=(nonesuch const&) = delete;
};
```

4 D > 4 D > 4 E > 4 E > E + 2 C C

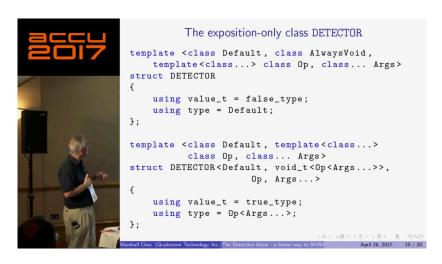


Figure 3: Inline 70%

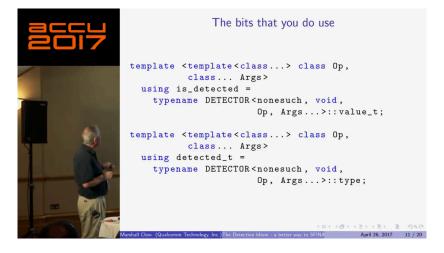


Figure 4: Inline 70%



How does it work?

It is surprisingly simple. You provide a 'type function'; a template that takes a type (or series of types) and returns a type.

Then you use is_detected to apply that type function to the types that you're interested in, and it returns to you: (a) whether or not this worked, and (b) if it did, what the result (type) was.

4 D > 4 D > 4 Z > 4 Z > Z + 9 Q C

Figure 5: Inline 70%



A slightly more practical example

```
struct Yes { typedef size_t size_type; };
struct No { /* no 'size_type' */ };

template <typename T>
    using has_size = typename T::size_type;

template <typename T>
void doSomething( const T& t )
{
    detected_or_t < short, has_size, T> v = 123;
// more with v
}
```

Figure 6: Inline 70%



Figure 7: Inline 70%

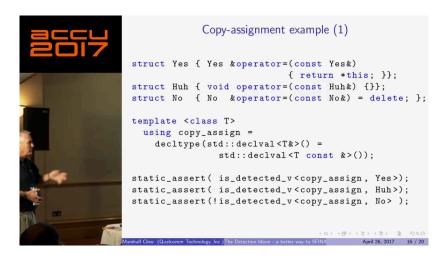


Figure 8: Inline 70%

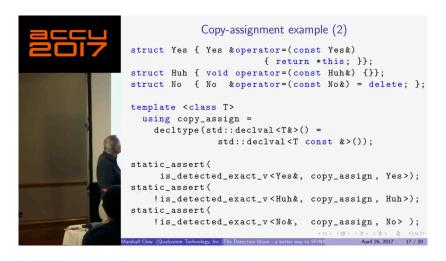


Figure 9: Inline 70%

pystring by Sony ImageWorks

GitHub

Pystring is a collection of C++ functions which match the interface and behavior of python's string class methods using std::string. Implemented in C++, it does not require or make use of a python interpreter. It provides convenience and familiarity for common string operations not included in the standard C++ library. It's also useful in environments where both C++ and python are used.

Overlapping functionality (such as index and slice/substr) of std::string is included to match python interfaces.

Modern C++ Design reloaded - Odin Holmes [ACCU 2017]

Video

- This is an "update" to the book "Modern C++ Design" by Andrei Alexandrescu
- Discusses updated template metaprogramming techniques:
 - policy-based class design;
 - replacing multiple templates with variadic templates.

Modern C++ Design reloaded (cont.)

"Concept"-based polymorphism (1)

```
class drawable concept{
    drawable concept() = default;
    virtual ~drawable concept() = default;
    virtual void draw() = 0;
   T model ;
    drawable model(T const& model) : model (model){}
    void dra\overline{w}() {
        model .draw();
    ~drawable model() = default;
```

Figure 10: 150%

Modern C++ Design reloaded (cont.)

"Concept"-based polymorphism (2)

```
class drawable{
    std::unique ptr<drawable concept> object ;
    public:
    template <typename T>
    drawable (const T& x) : object (
        new drawable model<T>(x))
    void draw() {
        object ->draw();
struct my widget{
    void \overline{d}raw(){};
.nt main(){
```

Bjarne Stroustrup on Mapping the Journey podcast

Transcript

The Guidelines Support Library - Rainer Grimm

Post

- ▶ Microsoft GSL
- ▶ GSL-Lite works with C++98 and C++03
- Quick overview of the classes available

The Ultimate Question of Programming, Refactoring, and Everything

by Andrey Karpov, PVS Studio

Post

- Tips and tricks on C++ programming
- Examples of bugs and bad practices
- Lots of code snippets
- Based on PVS-Studio diagnostics

&& == and, & == bitand





Of *course* you can use the C++ `and` and `bitand` keywords as reference qualifiers

```
struct RValue {
   RValue(const RValue bitand);
};

RValue and rvalue(RValue and) and;
```

5:30 PM - 25 Jul 2017

19 Retweets 44 Likes



Figure 12: 70%

STL on friends

●●●●○ O2-UK 4G

17:33

→ 100% **-**



Detail





Stephan T. Lavavej @StephanTLavavej What's the most complicated keyword in C++?

Small brain: class

Normal brain: template

Big brain: double

Galactic brain: friend



106 Likes

26 Retweets