

C++ Club UK

Gleb Dolgich

2019-01-17

Intel Contributes Its Parallel STL Implementation To LLVM

- ▶ Intel Contributes Its Parallel STL Implementation To LLVM
- ▶ Announcement:
<https://lists.llvm.org/pipermail/cfe-dev/2018-December/060606.html>
- ▶ Code: <https://github.com/llvm-mirror/pstl/commits/master>
- ▶ Reddit: https://www.reddit.com/r/cpp/comments/a9n0gk/intel_contributes_its_parallel_stl_implementation/

Oh DeaR

- ▶ Oh DeaR by JeanHeyd Meneide
- ▶ Reddit: https://www.reddit.com/r/cpp/comments/a5rkfr/oh_dear/

Rant alert: C++ error handling

Reddit: https://www.reddit.com/r/cpp/comments/ae60nb/decades_have_passed_standard_c_has_no_agreed_and/

I think the problem really stems from the places where exceptions don't work well. Otherwise I'd use them all the time. <...> E.g. im working on an app using actors which send messages back and forth. Throwing an exception in the message handler may mean something but it certainly isn't going to automatically make its way back to it's source.¹

FYI Outcome and std::expected<T, E> have diverged by a fair bit <...>. They are now two differing interpretations of how best to implement exception-less failure handling. There is a third approach proposed for Boost as well, called LEAF.²

¹https://www.reddit.com/r/cpp/comments/ae60nb/decades_have_passed_standard_c_has_no_agreed_and/edmnpez/

²https://www.reddit.com/r/cpp/comments/ae60nb/decades_have_passed_standard_c_has_no_agreed_and/edmpcon/

Game Programmers and the C++ collapse

[Game Programmers and the C++ collapse](#) by Alexis Breust

Let's focus on why C++ is slowly digging his own grave.

C++ is too slow to compile.

Rainbow Six: Siege (8 millions C++ LOC) in 3 minutes. That's extremely slow!

Fact is Modern C++ is just growing up so much it will soon collapse under its own mess.

Gamedev C++ coding decisions (in 2014)

Nicholas Fleury, CppCon 2014:

- ▶ No RTTI: you prefer to control the memory details yourself.
- ▶ No exceptions: they are simply slow because of the constraints of unwindable stack at any time.
- ▶ No STL containers: std::vector is slow, and complex template code takes time to compile.
- ▶ No Boost: surely the library is impressive, but it's never a good idea to use it in serious applications.
- ▶ Very small subset of templates: use it at minimum as it increases compile times significantly.

New languages for gamedev:

- ▶ [Jai](#) (a better C) by Jonathan Blow (still unreleased)
- ▶ [Odin](#) (based on Jai)

Orthodox C++ (!)

- ▶ [Orthodox C++](#) by Branimir Karadzic

Orthodox C++ (sometimes referred as C+) is minimal subset of C++ that improves C, but avoids all unnecessary things from so called Modern C++. It's exactly opposite of what Modern C++ is supposed to be.

Orthodox C++: what should I use?

► Branimir Karadzic:

- ▶ C-like C++ is good start, if code doesn't require more complexity don't add unnecessary C++ complexities. In general case code should be readable to anyone who is familiar with C language.
- ▶ The design rationale in Orthodox C++ should be "Quite simple, and it is usable. EOF".
- ▶ Don't use exceptions.
- ▶ Don't use RTTI.
- ▶ Don't use C++ runtime wrapper for C runtime includes (<cstdio>, <cmath>, etc.), use C runtime instead (<stdio.h>, <math.h>, etc.)
- ▶ Don't use stream (<iostream>, <stringstream>, etc.), use printf style functions instead.
- ▶ Don't use anything from STL that allocates memory, unless you don't care about memory management.
- ▶ Don't use metaprogramming excessively. Use it in moderation, only where necessary, and where it reduces code complexity.
- ▶ Wary of any features introduced in current standard C++, ideally wait for improvements of those feature in next iteration of standard. Example: `constexpr` from C++11 became usable in C++14 (Jason Turner)

Orthodox C++: Other efforts

- ▶ Nominal C++ by Naman Dixit, 2016
- ▶ Sane C++ by Andre "Floh" Weissflog, 2013
- ▶ Why Your C++ Should Be Simple by Benjamin Supnik, 2017

Why I don't spend time with Modern C++ anymore

- ▶ Why I don't spend time with Modern C++ anymore by Henrique Bucher, ED, JP Morgan (!) ([Original](#))
 - ▶ Performance loss, optimization difficulties
 - ▶ Slow build times
 - ▶ Complexity and maintainability
- ▶ Quotes:
 - ▶ If you cannot figure out in one minute what a C++ file is doing, assume the code is incorrect.
 - ▶ C++ today is like Fortran: it reached its limits.
 - ▶ Today the “Modern Technologist” has to rely on a new set of languages: Verilog, VHDL.
 - ▶ Vitorian LLC – we help businesses design, architect and build ultra-low-latency systems (*ka-ching!*)
- ▶ Discussions:
 - ▶ https://www.reddit.com/r/programming/comments/4jsaxb/why_i_dont_spend_time_with_modern_c_anymore/
 - ▶ <https://news.ycombinator.com/item?id=11720659>

CppCon 2018: Stephan T. Lavavej “Class Template Argument Deduction for Everyone”

- ▶ Stephan T. Lavavej - Class Template Argument Deduction for Everyone
 - ▶ Slides: https://github.com/CppCon/CppCon2018/blob/master/Presentations/class_template_argument_deduction_for_everyone/class_template_argument_deduction_for_everyone_stephan_t_lavavej_cppcon_2018.pdf
 - ▶ Reddit: https://www.reddit.com/r/cpp/comments/9newpl/how_to_use_class_template_argument_deduction/
- ▶ How to Use Class Template Argument Deduction blog post by STL
- ▶ CppReference: https://en.cppreference.com/w/cpp/language/class_template_argument_deduction

CppCon 2018: Stephan T. Lavavej “Class Template Argument Deduction for Everyone”

cppcon | 2018

THE C++ CONFERENCE • BELLEVUE, WASHINGTON

9

CTAD Syntax

- Works with parentheses and braces:

```
pair x(11, 22);  
pair y{11, 22};
```

- Works with direct-init and copy-init:

```
pair z = y;  
pair a = { 11, 22 };
```

- Works with named variables and temporaries:

```
vector<UserDefinedType> v;  
v.emplace_back(pair(11, 22));  
v.emplace_back(pair{11, 22});
```



STEPHAN T. LAVAVEJ

Class Template
Argument Deduction
for Everyone

CppCon.org

CppCon 2018: Stephan T. Lavavej “Class Template Argument Deduction for Everyone”

The slide has a blue header bar with the number '10' in white. The main title 'What CTAD Is Doing' is in large, bold, dark blue font. Below it is a bulleted list of points about Class Template Argument Deduction (CTAD). On the right side of the slide, there is a video frame showing Stephan T. Lavavej speaking on stage. The video frame has a black border and is positioned next to a text box containing his name and the presentation title. At the bottom right of the slide, there is a yellow button-like element with the text 'CppCon.org'.

10

What CTAD Is Doing

- C++17 performs template argument deduction
 - When constructing an object
 - Given only the name of a class template
 - Constructor arguments provide type information
- Usual deduction rules and library authors control what types are deduced
 - 1729 is an rvalue of type int
 - "taxicab" is an lvalue of type const char [8]
 - The STL deduces pair<int, const char *>
 - We'll see how this works later (deduction guides)

STEPHAN T. LAVAVEJ

Class Template Argument Deduction for Everyone

CppCon.org

CppCon 2018: Stephan T. Lavavej “Class Template Argument Deduction for Everyone”

The screenshot shows a presentation slide from CppCon 2018. The slide has a blue header bar with the number '14' in white. Below the header, the title 'Reverse Sort' is displayed in large, bold, dark blue font. To the right of the title is a block of C++ code. On the far right of the slide, there is a portrait of Stephan T. Lavavej, a man with glasses and a beard, wearing a dark t-shirt, standing and gesturing. The background of the slide is dark blue with some abstract light patterns.

14

Reverse Sort

```
array arr = { "lion"sv, "direwolf"sv,  
    "stag"sv, "dragon"sv };  
sort(arr.begin(), arr.end(), greater{});  
cout << arr.size() << ":";  
for (const auto& e : arr) { cout << e << " "; }  
cout << "\n";  
// 4: stag lion dragon direwolf
```

STEPHAN T. LAVAVEJ

Class Template Argument Deduction for Everyone

CppCon.org

CppCon 2018: Stephan T. Lavavej “Class Template Argument Deduction for Everyone”

cppcon | 2018

THE C++ CONFERENCE • BELLEVUE, WASHINGTON

23

C++17 CTAD Limitations

- P1021 by Mike Spertus and Timur Doumler proposes fixing several C++17 CTAD limitations for C++20
- CTAD currently doesn't work with:
 - Alias templates
 - Explicit template arguments
- CTAD currently needs deduction guides for:
 - Aggregates (like `std::array`)
 - Inherited constructors



STEPHAN T. LAVAVEJ

Class Template
Argument Deduction
for Everyone

CppCon.org

CppCon 2018: Stephan T. Lavavej “Class Template Argument Deduction for Everyone”

cppcon | 2018

THE C++ CONFERENCE • BELLEVUE, WASHINGTON

27

Non-Templated Constructors

```
template <typename A, typename B>
struct MyPair {
    MyPair(const A&, const B&) { }
};

MyPair mp1{1729, 3.14}; // MyPair<int, double>
MyPair mp2{22, "meow"}; // MyPair<int, char[5]>

• Works automatically without deduction guides
  • If you don't want MyPair<int, char[5]>, you'll need guides
```



STEPHAN T. LAVAVEJ

Class Template
Argument Deduction
for Everyone

CppCon.org

CppCon 2018: Stephan T. Lavavej “Class Template Argument Deduction for Everyone”

The image shows a presentation slide from CppCon 2018. At the top right, the logo reads "cppcon | 2018 THE C++ CONFERENCE • BELLEVUE, WASHINGTON". Below the logo is a photograph of Stephan T. Lavavej, a man wearing sunglasses and a dark t-shirt, standing on a stage. To his right, a text box contains his name, "STEPHAN T. LAVAVEJ". The main content of the slide is a list under the heading "Will CTAD Automatically Work?". The list is divided into two sections: "CTAD automatically works when:" and "CTAD doesn't automatically work when:". Both sections contain three bullet points each.

Will CTAD Automatically Work?

- CTAD automatically works when:
 - A class template has a constructor whose signature mentions all of the class template's parameters
 - `MyPair<A, B>` had `MyPair(const A&, const B&)`
 - Or the class template provides default template arguments
 - `Spot<T, Alloc>` had `Spot(const T&)` and `Alloc = allocator<T>`
- CTAD doesn't automatically work when:
 - Class template parameters aren't mentioned/defaulted
 - Arguments prevent deduction (e.g. `nullptr` arg for `B *`)
 - Parameters are non-deducible (e.g. `list<T>::iterator`)

STEPHAN T. LAVAVEJ

Class Template Argument Deduction for Everyone

CppCon.org

CppCon 2018: Stephan T. Lavavej “Class Template Argument Deduction for Everyone”

cppcon | 2018

THE C++ CONFERENCE • BELLEVUE, WASHINGTON

36

Forwarding Constructors

```
template <typename A, typename B>
    struct AdvancedPair {
        template <typename T, typename U>
            AdvancedPair(T&&, U&&) { }

    };
    • Perfect forwarding inhibits CTAD
    • We want to imitate make_pair() which decays
    • Note: make_pair() unwraps reference_wrapper
        • std::pair CTAD doesn't; reference_wrapper is rare
```



STEPHAN T. LAVAVEJ

Class Template
Argument Deduction
for Everyone

CppCon.org

CppCon 2018: Stephan T. Lavavej “Class Template Argument Deduction for Everyone”

cppcon | 2018

THE C++ CONFERENCE • BELLEVUE, WASHINGTON

39

What "Decay by Value" Illustrates

- CTAD determines what type to construct
 - Input: constructor args, constructors, deduction guides
 - Performs template arg deduction and overload resolution
 - Output: a specific type (or deduction failure)
- Next, overload resolution happens again, normally
- **CTAD doesn't affect the constructor call**
 - CTAD and deduction guides don't affect existing code
- Example:
 - CTAD uses AdvancedPair(X, Y)
 - CTAD deduces AdvancedPair<int, const char *>
 - Overload resolution selects AdvancedPair(T&, U&&)



STEPHAN T. LAVAVEJ

Class Template
Argument Deduction
for Everyone

CppCon.org

CppCon 2018: Stephan T. Lavavej “Class Template Argument Deduction for Everyone”

cppcon | 2018
THE C++ CONFERENCE • BELLEVUE, WASHINGTON

41

Supporting CTAD in Your Library

- For each constructor of each class template:
 - Is CTAD applicable? (Example "no": `vector<T>(size_t)`)
 - Does CTAD automatically work?
 - And does it do what you want? (Recall `MyPair<int, char[5]>`)
 - Or do you need to write a deduction guide?
- If you support pre-C++17, guard your guides
 - Feature-test macro: `__cpp_deduction_guides`
- Write simple tests with `static_assert`
 - Preventing constructors/typedefs from breaking things



STEPHAN T. LAVAVEJ

Class Template
Argument Deduction
for Everyone

CppCon.org

CppCon 2018: Stephan T. Lavavej “Class Template Argument Deduction for Everyone”

cppcon | 2018

THE C++ CONFERENCE • BELLEVUE, WASHINGTON

52

tuple/optional Avoid Wrapping

```
tuple tup{11, 22, 33}; // tuple<int, int, int>
tuple tup2{tup};        // tuple<int, int, int>
```

```
optional opt{"meow"s}; // optional<string>
optional opt2{opt};    // optional<string>
```

- CTAD prioritizes everyday code
- Expert library authors need to be aware of this
 - If they want tuple<tuple<Args>>, optional<optional<T>>



STEPHAN T. LAVAVEJ

Class Template
Argument Deduction
for Everyone

CppCon.org

CppCon 2018: Stephan T. Lavavej “Class Template Argument Deduction for Everyone”

cppcon | 2018

THE C++ CONFERENCE • BELLEVUE, WASHINGTON

53

initializer_list Overloading

- Examples mentioned by Nicolai Josuttis

```
set<int> s;
vector v1(s.begin(), s.end()); // vector<int>
vector v2{s.begin(), s.end()};
// vector<set<int>::iterator>
```

```
vector v3{"ab", "cd"}; // vector<const char *>
vector v4("ab", "cd"); // vector<char>, UB!
```

- Unlikely to be problematic in everyday code



STEPHAN T. LAVAVEJ

Class Template
Argument Deduction
for Everyone

CppCon.org

CppCon 2018: Stephan T. Lavavej “Class Template Argument Deduction for Everyone”

The image shows a presentation slide from CppCon 2018. The slide has a blue header bar with the number '61' in white. The main title is 'Fixing basic_string CTAD'. Below the title is a bulleted list of five items. To the right of the slide, there is a video frame showing Stephan T. Lavavej speaking on stage. The video frame has a black border and includes the CppCon logo at the top. At the bottom of the video frame, the speaker's name 'STEPHAN T. LAVAVEJ' is displayed. Below the video frame, the title 'Class Template Argument Deduction for Everyone' is shown. At the very bottom of the slide, the CppCon website 'CppCon.org' is listed.

61

Fixing `basic_string` CTAD

- `basic_string`'s ctors are too heavily overloaded
- In these cases, CTAD deduced `Allocator = int`
- Deduction guides can't easily fix this scenario
- Fix: constrain `Allocator` to be an allocator
 - Avoids affecting actual construction
 - Probably superseded by concepts

STEPHAN T. LAVAVEJ

Class Template Argument Deduction for Everyone

CppCon.org

Returning values in parentheses



Walletfox
@walletfox FOLLOW YOU

To clarify: this only happens if you return decltype (auto) and use parentheses - you will get a dangling reference.
If you return auto, there is no issue with return value in parentheses.

wandbox.org/permlink/4CpsW... #cpp14
pic.twitter.com/Gtv8WB6G3U

```
// ok, returns int
auto f()
{
    int a = 4;
    return (a);
}

// warning: reference to local variable 'a'
// returns int&
decltype(auto) g()
{
    int a = 4;
    return (a);
}
```

9 Likes 0 Retweets

8 Jan 2019 at 09:02 via Twitter Web Client

Quote

The 'S' in IoT stands for 'Security'.

Twitter

 **Marco Arena**
@ilpropheta

When C++ programmers find undefined behaviour in other languages.
#cpp #humour #programming pic.twitter.com/mDooUft4ly



Chewie, we're home

56 Likes	16 Retweets
14 Dec 2018 at 09:12	via Twitter for Android