

C++ Club Meeting Notes

Gleb Dolgich

2018-12-20

Guaranteed Copy Elision Does Not Elide Copies

VCBlog post by Simon Brand :: Reddit

Continuing the theme of C++ misnomers, value categories are not categories of values; they are characteristics of expressions.

1. A *glvalue* (generalized lvalue) is an expression whose evaluation determines the identity of an object, bit-field, or function.
2. A *prvalue* is an expression whose evaluation initializes an object or a bit-field, or computes the value of an operand of an operator, as specified by the context in which it appears.
3. An *xvalue* is a glvalue that denotes an object or bit-field whose resources can be reused (usually because it is near the end of its lifetime).
4. An *lvalue* is a glvalue that is not an xvalue.
5. An *rvalue* is a prvalue or an xvalue.

That's a better name for this feature. Not guaranteed copy elision.

Deferred temporary materialization.

CppCon 2018: Victor Ciura: Enough string_view to Hang Ourselves (1/8)

YouTube

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

VICTOR CIURA

Enough string_view
to Hang Ourselves

Convenience Conversions (and Gotchas)

- `const char *` automatically converts to `std::string` via constructor (*not explicit*)
- `const char *` automatically converts to `std::string_view` via constructor (*not explicit*)
- `std::string` automatically converts to `std::string_view` via conversion operator
- can construct a `std::string` from a `std::string_view` via constructor (*explicit*)

2018 Victor Ciura 39

CppCon.org

CppCon 2018: Victor Ciura: Enough string_view to Hang Ourselves (2/8)



The slide features a video frame of Victor Ciura, a man with glasses and a pink shirt, gesturing while speaking. Below the video frame is his name, VICTOR CIURA.

string and string_view

⚠ Don't use `std::string_view` to initialize a `std::string` member !

If you know that you're ultimately going to create a `std::string`
make the whole *call chain* use `std::string`
don't mix-in `std::string_view` along the way.

🚫 `string_view` → `string` → `string_view` → ... → `std::string`
(maybe a string literal)

2018 Victor Ciura 49

Enough string_view
to Hang Ourselves

CppCon.org

CppCon 2018: Victor Ciura: Enough string_view to Hang Ourselves (3/8)

Lifetime profile v1.0

<https://herbsutter.com/2018/09/20/lifetime-profile-v1-0-posted/>

```
void example_2_6_2_1()
{
    std::string_view s = "foo"s;      // A
    s[0];   // ERROR (lifetime.3): 's' was invalidated when
           // temporary "foo"s' was destroyed (line A)
}
```

 CppCoreGuidelines

<https://github.com/isocpp/CppCoreGuidelines/blob/master/docs/Lifetime.pdf>

2018 Victor Ciura 58

VICTOR CIURA

**Enough string_view
to Hang Ourselves**

CppCon.org

CppCon 2018: Victor Ciura: Enough string_view to Hang Ourselves (4/8)

The slide is titled "Lifetime profile v1.0" and features a yellow star icon with the word "NEW" below it. It includes a screenshot of Victor Ciura speaking at a podium, a link to his blog post, and a code example demonstrating a clang warning about dangling pointers. The CppCoreGuidelines logo is present, along with a GitHub link to the guidelines document.

Lifetime profile v1.0

<https://herbsutter.com/2018/09/20/lifetime-profile-v1-0-posted/>

```
<source>:7:5: warning: passing a dangling pointer as argument [-Wlifetime]
    s[0];                                // ERROR (lifetime.3): 's' was invalidated when
    ^
<source>:6:32: note: temporary was destroyed at the end of the full expression
    std::string_view s = "foo"s;           // A
    ^
1 warning generated.
Compiler returned: 0
```

clang -Wlifetime

<https://github.com/isocpp/CppCoreGuidelines/blob/master/docs/Lifetime.pdf>

2018 Victor Ciura 59

VICTOR CIURA

**Enough string_view
to Hang Ourselves**

CppCon.org

CppCon 2018: Victor Ciura: Enough string_view to Hang Ourselves (5/8)

cppcon | 2018

THE C++ CONFERENCE • BELLEVUE, WASHINGTON



VICTOR CIURA

Enough string_view
to Hang Ourselves

std::string_view cheatsheet

Lifetime with std::string_view (C++17)
std::string_view isn't a drop-in replacement
for const std::string&

```
std::string str() {  
    return std::string("long_string_helps_to_detect_issues");  
}
```

const std::string& s = str();
std::cout << s << '\n';
lifetime extended
prints the correct result ✓

std::string_view sv = str();
std::cout << sv << '\n';
lifetime not extended
prints nonsense X

const lvalue reference binds to rvalue and provides lifetime extension. But there is no lifetime extension for std::string_view.

For short strings this issue might be hard to detect due to short string optimization (SSO). The problem becomes obvious with longer (dynamically allocated) strings.

@walletfox

2018 Victor Ciura

60

CppCon.org

CppCon 2018: Victor Ciura: Enough string_view to Hang Ourselves (6/8)

std::string_view is a borrow type

Borrow types are essentially “borrowed” references to existing objects.

- they lack ownership
- they are *short-lived*
- they generally can do without an *assignment operator*
- they generally appear only in *function parameter lists*
- they generally *cannot be stored in data structures or returned safely* from functions (no ownership semantics)

<https://quuxplusone.github.io/blog/2018/03/27/string-view-is-a-borrow-type/>

2018 Victor Ciura 87

VICTOR CIURA

**Enough string_view
to Hang Ourselves**

CppCon.org

CppCon 2018: Victor Ciura: Enough string_view to Hang Ourselves (7/8)

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

VICTOR CIURA

**Enough string_view
to Hang Ourselves**

std::string_view is a borrow type

`string_view` is perhaps the first “mainstream” **borrow type**.

BUT:

`string_view` is **assignable**: `sv1 = sv2`

Assignment has **shallow** semantics (of course, the viewed strings are *immutable*).

Meanwhile, the comparison `sv1 == sv2` has **deep** semantics.

<https://quuxplusone.github.io/blog/2018/03/27/string-view-is-a-borrow-type/>

2018 Victor Ciura 89

CppCon.org

CppCon 2018: Victor Ciura: Enough string_view to Hang Ourselves (8/8)

Simple rules for borrow types

Borrow types must appear **only as function parameters** and **for-loop control variables**.

We can make an **exception** for function **return types**:

- a function may have a borrow type as its return type
(the function must be **explicitly annotated** as returning a potentially dangling reference)
- the result returned **must not be stored** into any named variable,
except a function parameter or for-loop control variable

<https://quuxplusone.github.io/blog/2018/03/27/string-view-is-a-borrow-type/>

2018 Victor Ciura 96

VICTOR CIURA

**Enough string_view
to Hang Ourselves**

CppCon.org

MSVC class layout (1/3)



Timothy Lochner
@tloch14

One of the best, and least known features of MSVC, is /d1reportAllClassLayout

Even works in Godbolt: [godbolt.org/z/
PAZzx1](https://godbolt.org/z/PAZzx1)

(thanks [@rob_brink](#) for the continual reminder that it exists) [pic.twitter.com/
eYeQfVnKf1](https://pic.twitter.com/eYeQfVnKf1)

```
1 struct test
2 {
3     int blah;
4     float blah2;
5     float expectPaddingAfterThis;
6     float* lolcakes;
7     int* anotherPointer;
8 };
9
```

MSVC class layout (2/3)



Timothy Lochner
@tloch14

One of the best, and least known features of MSVC, is /d1reportAllClassLayout

Even works in Godbolt: [godbolt.org/z/
PAZzxI](https://godbolt.org/z/PAZzxI)

(thanks [@rob_brink](#) for the continual reminder that it exists) [pic.twitter.com/
eYeQfVnKf1](https://pic.twitter.com/eYeQfVnKf1)

```
class test      size(32):
    +---
0     | blah
4     | blah2
8     | expectPaddingAfterThis
     | <alignment member> (size=4)
16    | lolcakes
24    | anotherPointer
    +---
```

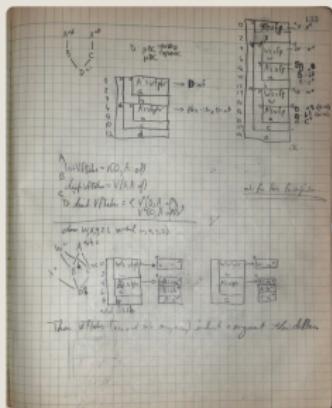
MSVC class layout (3/3)



Jan Gray @jangray

13h

@tloch14 @kenpex @rob_brink V0
(09/90) 😊



Simon Zeni @Bl4ckb0ne

9h

@tloch14 @rob_brink The same works
on linux with pahole!
godbolt.org/z/jPeCM2



Andrew Gresyk @andrew_gres...
@tloch14 @rob_brink + /
d1reportSingleClassLayout<name>

Conan, vcpkg or build2?

Reddit

- ▶ Pragmatic choice: vcpkg or Conan (they work today and are complete enough)
- ▶ Pragmatic no-brainer choice: vcpkg (it's the simplest and it have more packages ready)
- ▶ Pragmatic but need finer control choice: Conan (it gives more options)
- ▶ (Very) Long term choice: Build2 (shows great promises because it uses a coherent model...)
- ▶ Ideal choice (from the future): help SG15 (the group reflecting on tools vs C++) define interfaces for build systems and dependency managers so that your choice is not impacted by your dependencies choices.

Improving C++ Builds with Split DWARF

Article

```
1 | $ g++ -c -g -fPIC -fPIC main.cpp -o main.o  
2 | $ g++ main.o -o app
```

Having some fun with higher-order functions

- ▶ Article by Barry Revzin
- ▶ Boost.HOF

Compile-time raytracer by Tristan Brindle

- ▶ [Code](#)
- ▶ [Reddit](#)

Iterators: What Must Be Done?

- ▶ Article

Google C++ Style Guide is No Good

- ▶ Article by Eugene Yakubovich

Unlike C++ Core Guidelines that try to explain how to use the language effectively, GSG is about forbidding the use of certain features.

- ▶ Reddit

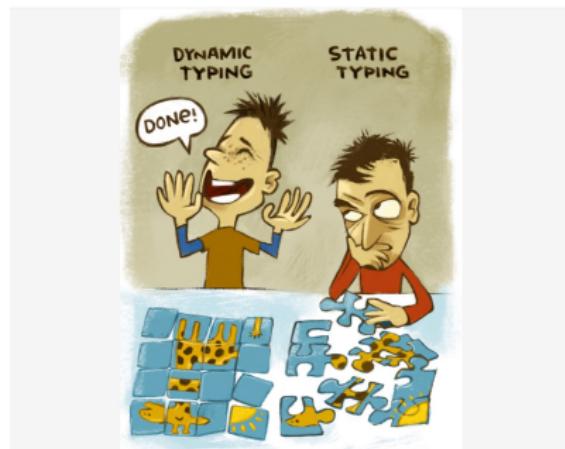
*There are issues with the google c++ style guide, but this article is bad.
It is basically just finding contrived cases where you have to use
questionable constructs, or willfully misinterpreting the document.*

Twitter



Kolja Wilcke
@01k

static vs dynamic #illustration
pic.twitter.com/nDgR8RGdPq



3,198 Likes

1,543 Retweets

28 Nov 2018 at 14:31

via Twitter Lite

Twitter



Dmitry Sviridkin
@Nekrolm

C85 — Implicit classes
C++98 — Attack of STLs
C++05 — Revenge of auto_ptr
C++11 — The new compilation error
C++14 — 'auto' strikes back
C++17 — Return of templates
C++20 — Modules awaken
C++23 — The last header

141 Likes 38 Retweets

11 Dec 2018 at 17:26 via Twitter for Android

Reply Retweet Like Share More

 **Eugene** @thedzhon
@Nekrolm C11 — Rogue One

1d