accu 2022

C++ MYTHBUSTERS

VICTOR CIURA

ACCU 2022

@ciura_victor

Victor Ciura
Principal Engineer

CAPHYON

Abstract

The C++ community is very large and quite vocal when it comes to controversial issues.
We're very fragmented on many topics, based on the breadth of the C++ ecosystem and the background/ experience we each bring from our C++ niche.

From CppCoreGuidelines to opinionated best practices to established idioms, there's a lot of good information easily available. Mixed up with all of this there are also plenty of myths. Some myths stem from obsolete information, some from bad teaching materials.

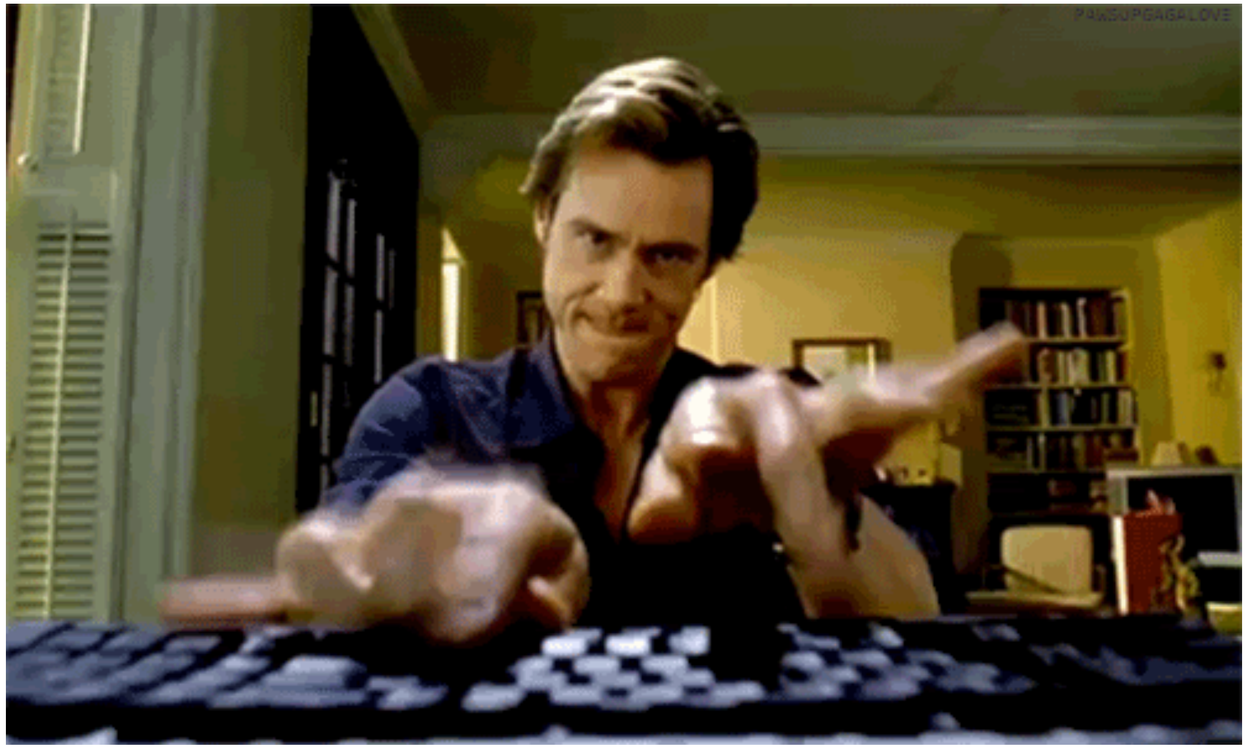
In this presentation, I will dissect a few of the most popular C++ myths to a level of detail not possible on Twitter… and without the stigma of newb/duplicate/eyeroll one might experience when asking these questions on StackOverflow.

Expect the familiar "Busted", "Plausible", or "Confirmed" verdicts on each myth and come prepared to chat about these.

Q & A

Do ask questions as we go along

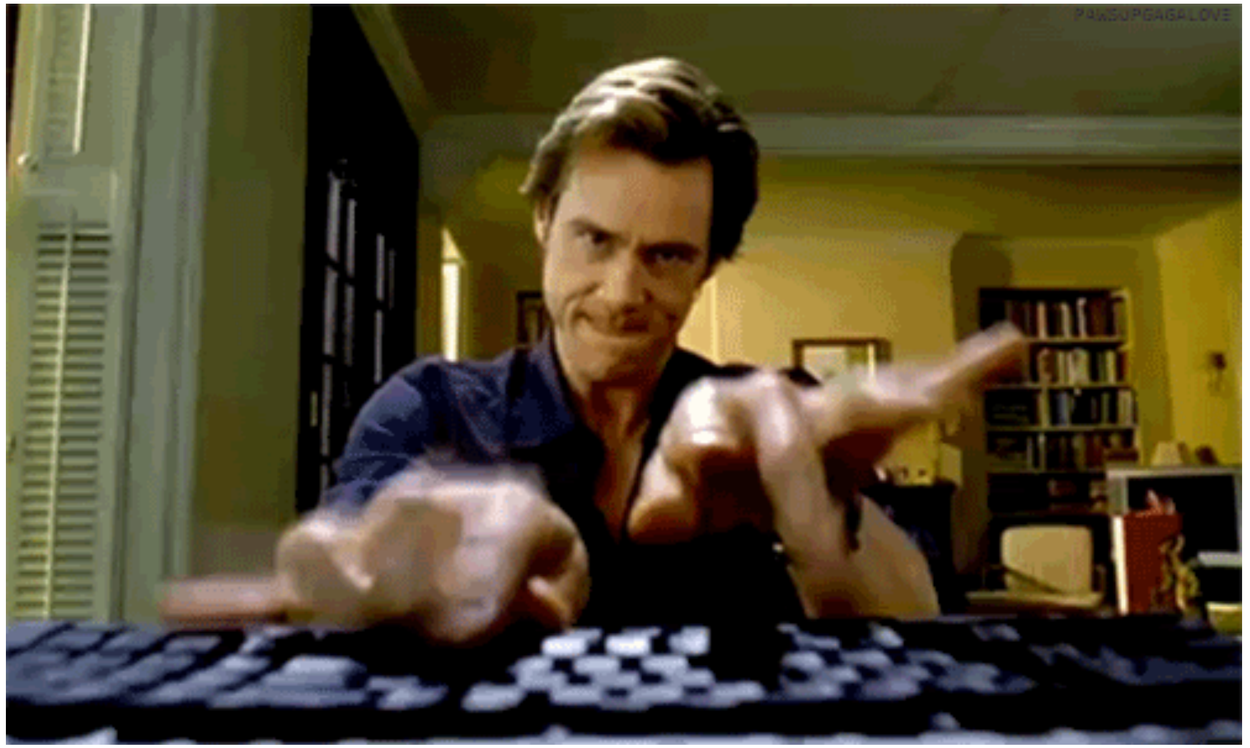
Comments are welcomed, too



Q & A

Do ask questions as we go along

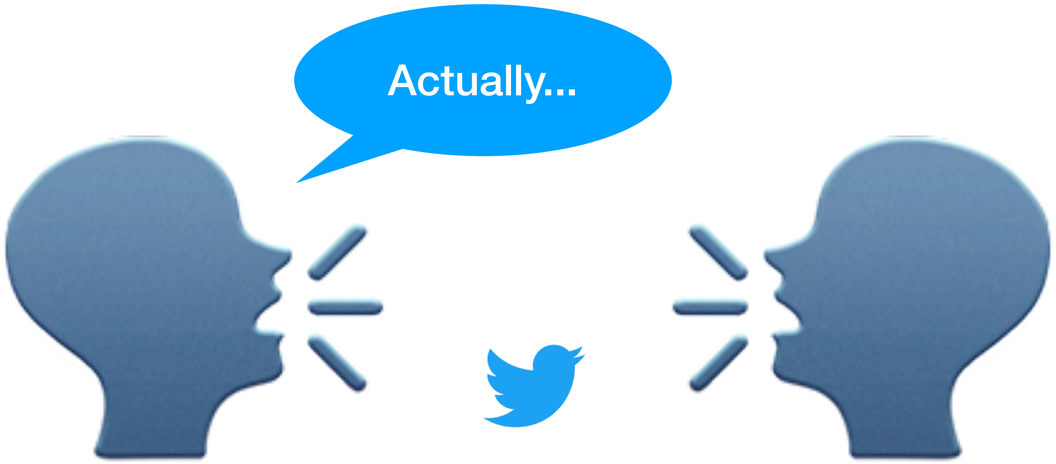
Comments are welcomed, too



Actually, ...

The C++ community is very large and quite vocal

when it comes to controversial issues



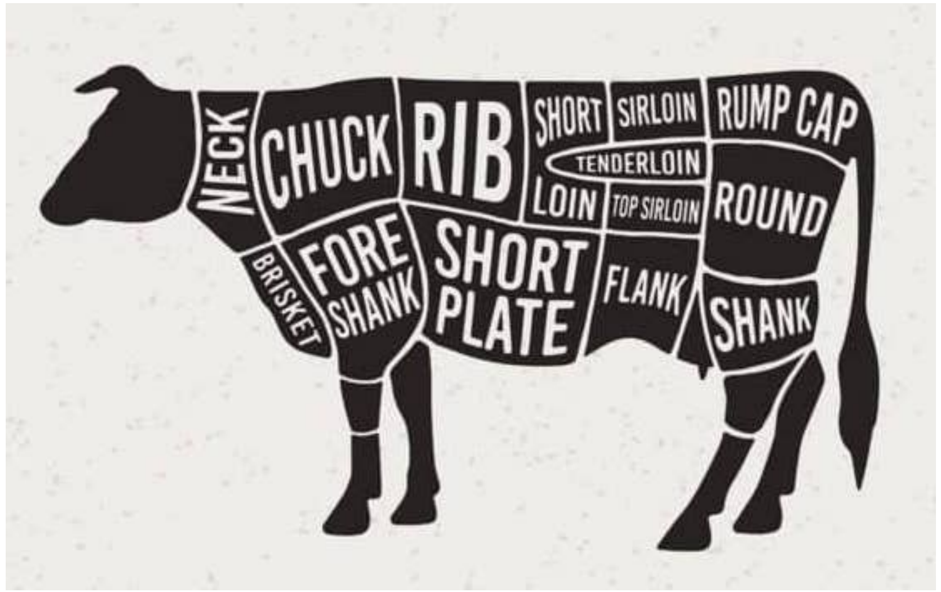
We're Different

We’re very fragmented on many topics

- based on the breadth of the C++ ecosystem
- background/experience we each bring from our C++ niche

We're Different

We’re very fragmented on many topics (Bjarne Stroustrup's 🐘 elephant metaphor)



Sources

A lot of good information easily available:

- CppCoreGuidelines
- (opinionated) best practices
- established idioms
- books
- conference presentations
- StackOverflow

🔥 Hot take typing

If it looks like a hot take, if it feels like a hot take... it probably is 😈

Myths

Mixed up with all of this, there are also plenty of myths

- some myths stem from obsolete information
- some from bad teaching materials

Let's talk about teaching C++

Not just about code reviews

Let's talk about teaching C++

Note that a few items might seem silly/obvious to you, but I've heard them and had to discuss them with junior devs or interns we hired fresh out of college.

Or questions I usually get when lecturing at my alma mater.

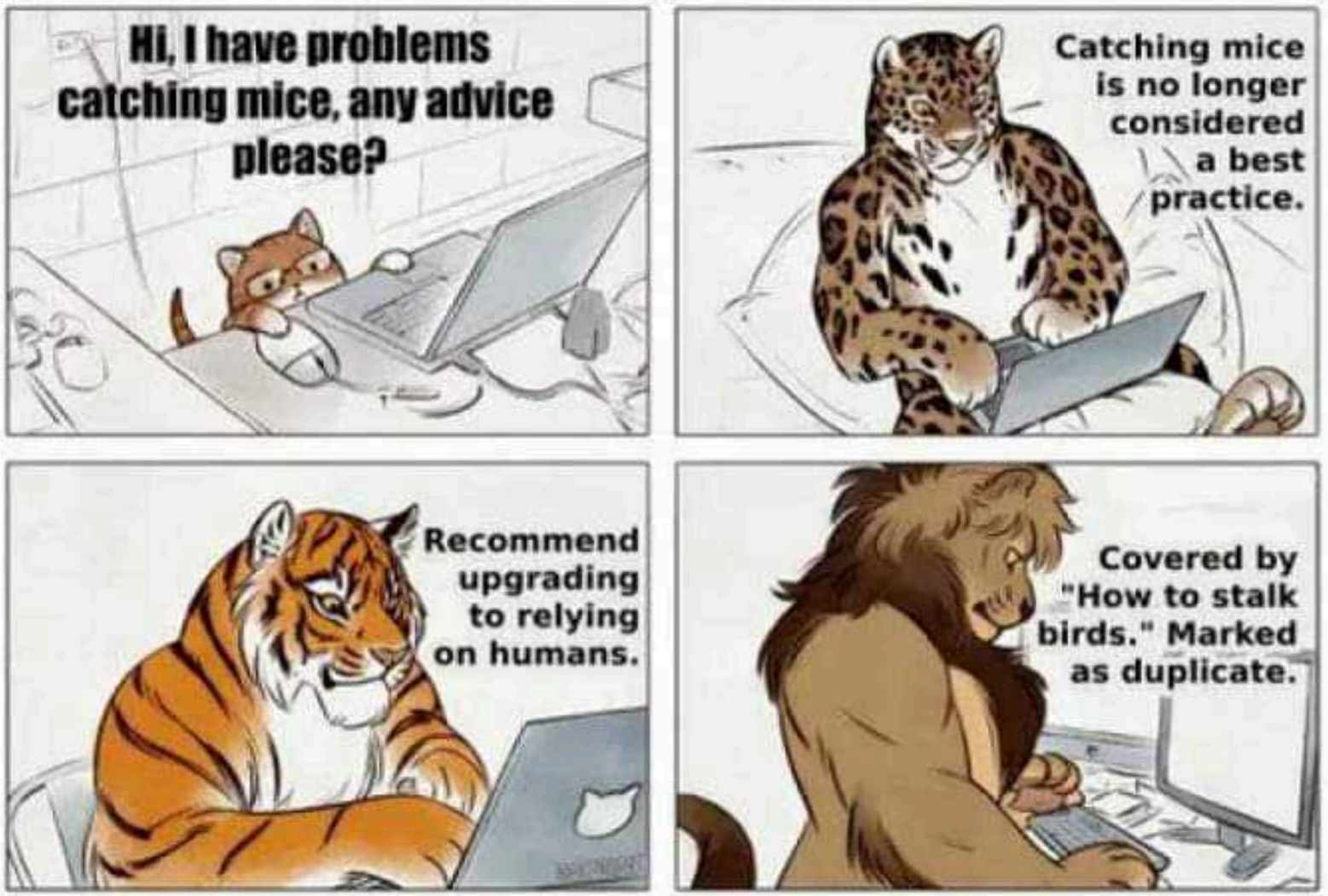
Not just about code reviews

#Programming #Cpp #AccuConf

C++ UNIverse - Victor Ciura [ACCU 2021]

youtube.com/watch?v=q0NJDr5hIWA

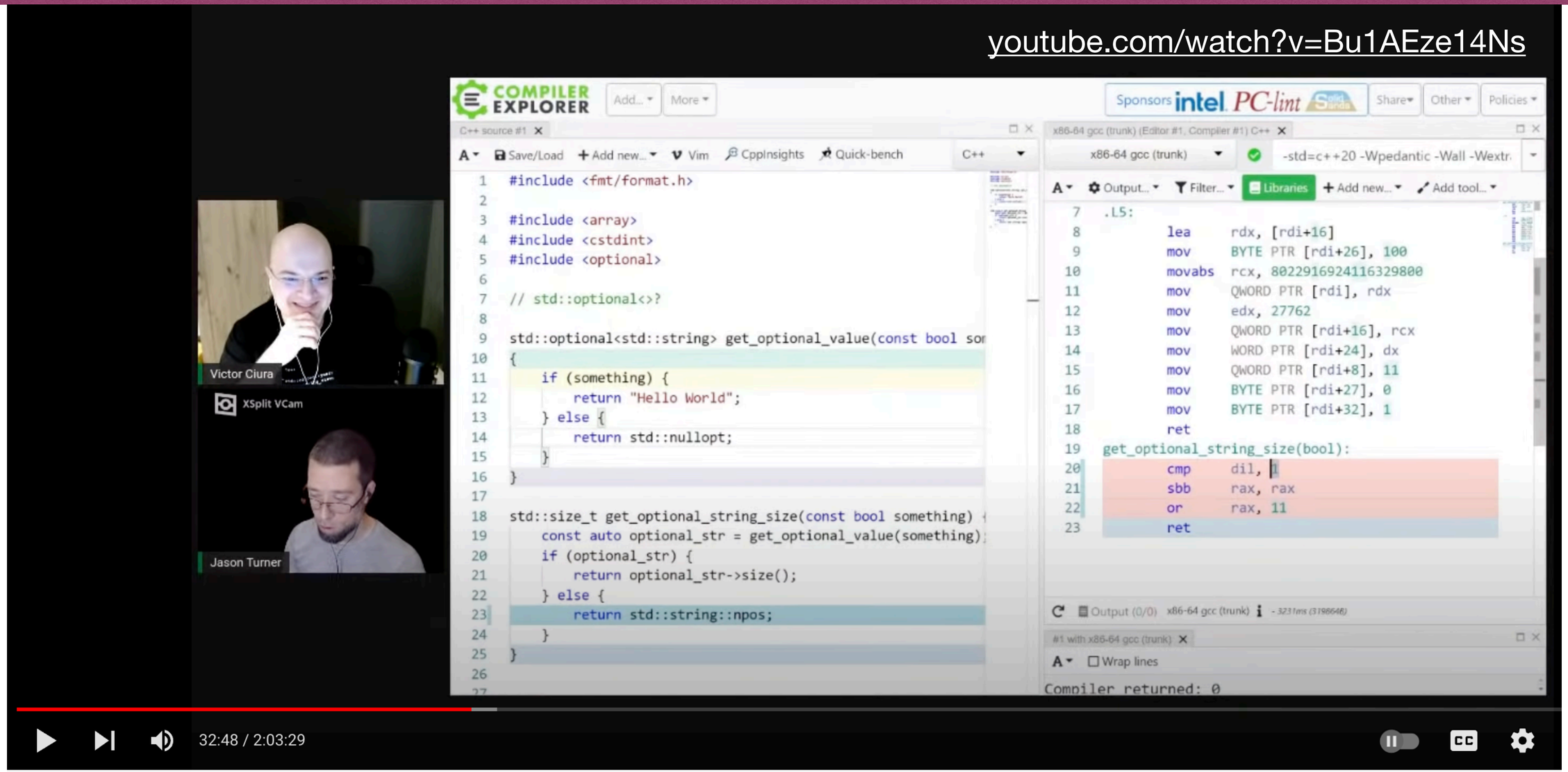
Help 😺



StackOverflow

Mythbusting with Jason - unscripted improv

youtube.com/watch?v=Bu1AEze14Ns



C++ Mythbusting with Victor and Jason

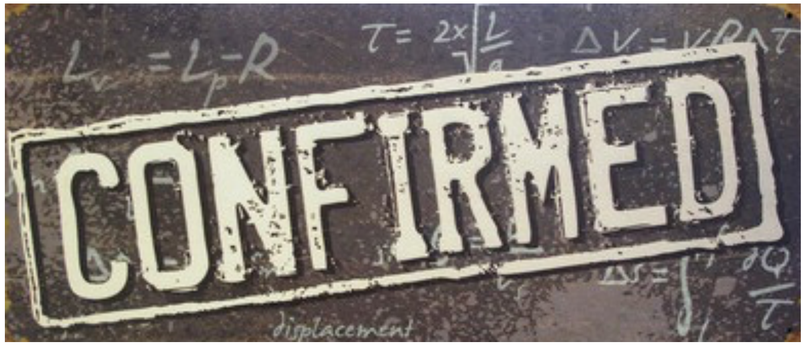
18,218 views • Streamed live on Jan 29, 2021

566 DISLIKE SHARE DOWNLOAD CLIP SAVE ...

Top chat replay

for templates, I would like to require

Verdict

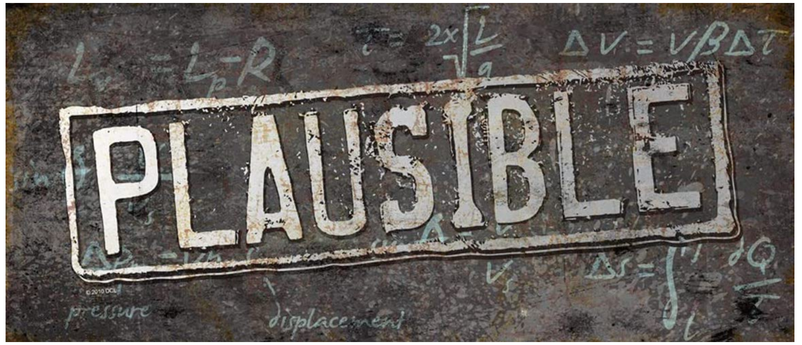


I want to instigate a healthy dialog, so speak up

Verdict

A programmer's staple response:

"It depends..." 🤓

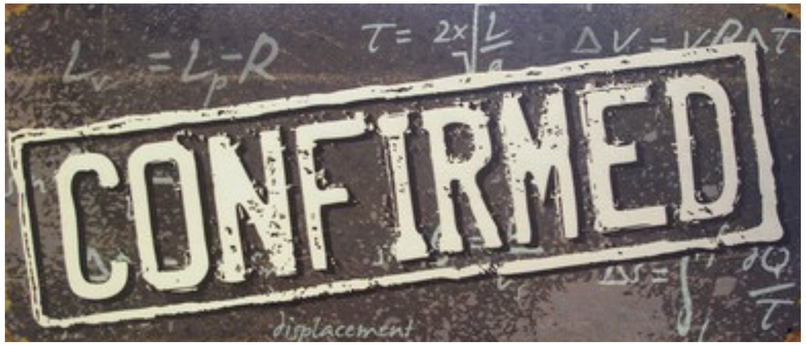


Verdict

Let's test this...

Myth

`iostreams` are slow



Just kidding 😄

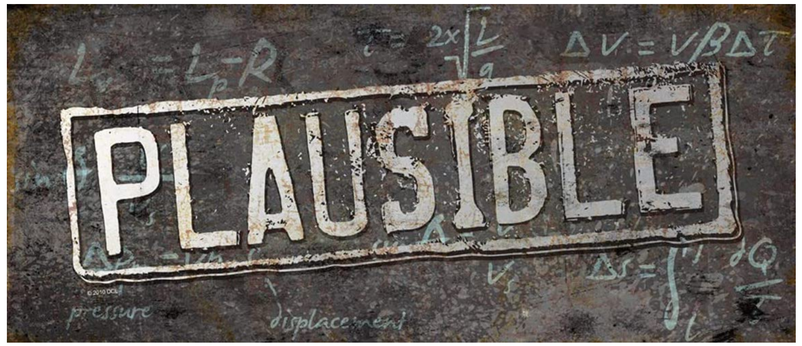
It's not a myth, we've known this for years.

Myth

C++20 modules will solve all problems

Myth

coroutines shipped in C++20



No library support 😔

We're going to get a generators library in C++23

Mythbusting Series

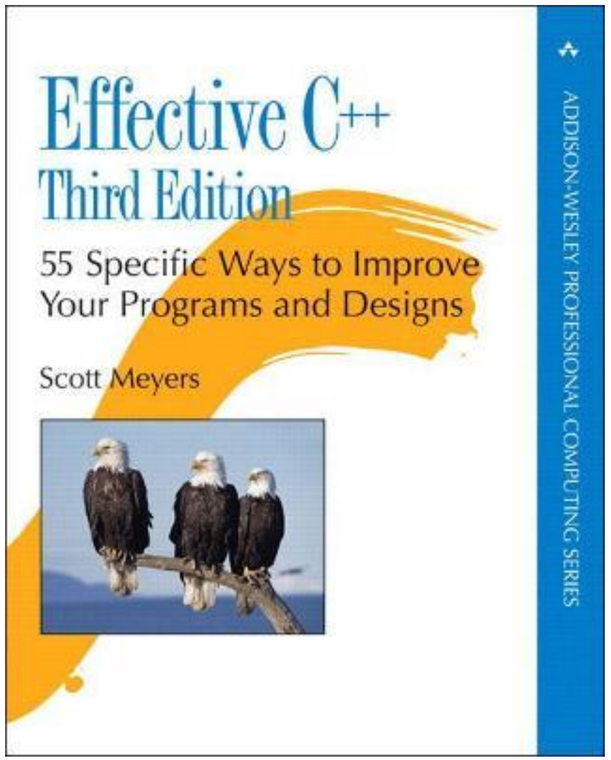
<Part 1 of N> presentation

Mythbusting Series

<Part 1 of N> presentation

I invite other speakers to **join** this C++ Mythbusting series

Scott Meyers indexing



The upshot of all this is simple. If you're dynamically allocating arrays, you're probably taking on more work than you need to. To lighten your load, use vectors or strings instead.

Item 14: Use reserve to avoid unnecessary reallocations.

One of the most marvelous things about STL containers is that they automatically grow to accommodate as much data as you put into them, provided only that you don't exceed their maximum size. (To discover this maximum, just call the aptly named max_size member function.) For vector and string, growth is handled by doing the moral equivalent of a realloc whenever more space is needed. This realloc-like

GotW #

Myth #11

`printf/sprintf` are very fast

Myth #11

`printf/sprintf` are very fast

`sprintf` uses the global locale

=> mutex lock ☹️

Myth #11

`printf/sprintf` are very fast

`sprintf` uses the global locale

=> mutex lock ☹️

On macOS, `sprintf` - that is in system libraries

ends up spending almost all the time inside a locale-related mutex lock 🔥

Myth #11

`printf/sprintf` are very fast

Blender case study: `sprintf => {fmt}`

- on macOS: 3-4**x** speedup
- on Windows: 20% speedup (due to a faster float/int formatter)

developer.blender.org/D13998

Myth #11

`printf/sprintf` are very fast

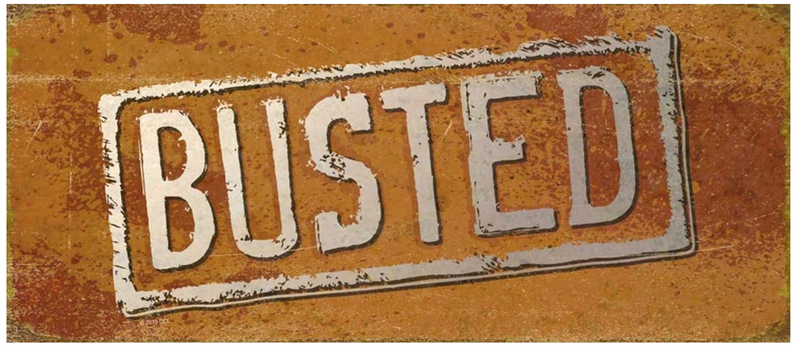
Blender case study: `sprintf => {fmt}`

- on macOS: 3-4**x** speedup
- on Windows: 20% speedup (due to a faster float/int formatter)

developer.blender.org/D13998

Myth #11

`printf/sprintf` are very fast



Myth #11

`printf/sprintf` are very fast

Use C++20 `std::format` or `{fmt}` library - `fmt::print()`

Myth #11

`printf/sprintf` are very fast

Use C++20 `std::format` or `{fmt}` library - `fmt::print()`

⚠ Beware of standard functions that use locale

Myth #14

C++ is not easily toolable 🛠️

Humans Depend on Tools

I'm a tool builder

[Advanced Installer]

[Clang Power Tools]

[caphyon.github.io]

Free/OSS 

Programmers Depend on Tools

good code editor
(or IDE)

linter/formatter

powerful (visual) debugger

(automated) refactoring tools

build system

package manager

SCM client

CI/CD service

code reviews platform

recent compiler(s)
[conformant/strict]

perf profiler

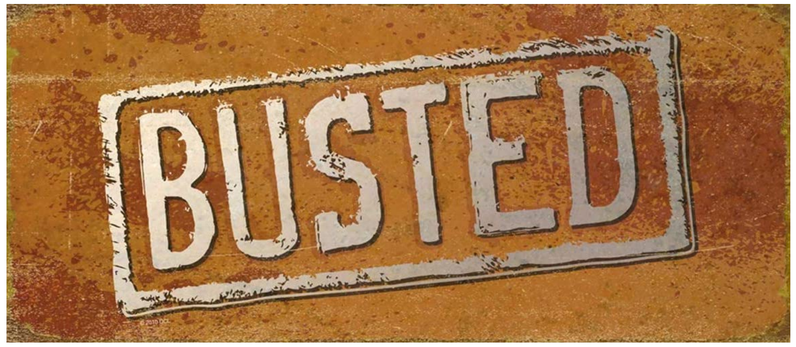
test framework

static analyzer

dynamic analyzer
(runtime)
+ fuzzing

Myth #14

C++ is not easily toolable 🛠️



Get to know your tools

well

Myth #19

`std::regex` is too slow for production use

```
const auto r = std::regex(R"((\S+)\s*=\s*(\S+))");

std::cmatch results;
const auto success = std::regex_match("x = 5", results, r);

fmt::print("Matched: {} '{}'='{}'", success,
           string(results[0].first, results[0].second),
           string(results[1].first, results[1].second));
```

Myth #19

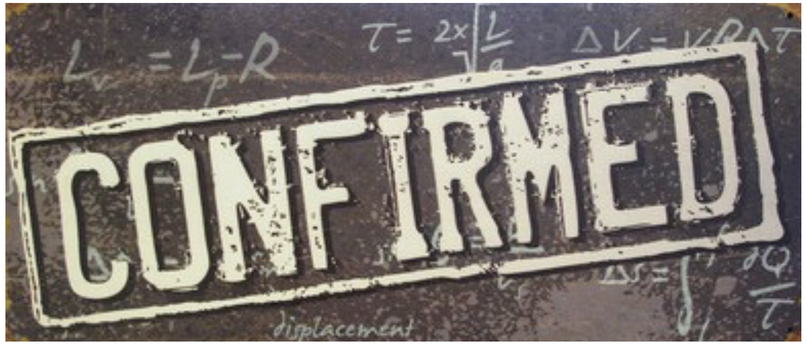
`std::regex` is too slow for production use

This short snippet is so slow to compile, it will actually timeout in CompilerExplorer 😄

```
const auto r = std::regex(R"((\S+)\s*=\s*(\S+))");

std::cmatch results;
const auto success = std::regex_match("x = 5", results, r);

fmt::print("Matched: {} '{}'='{}'", success,
           string(results[0].first, results[0].second),
           string(results[1].first, results[1].second));
```

Myth #19

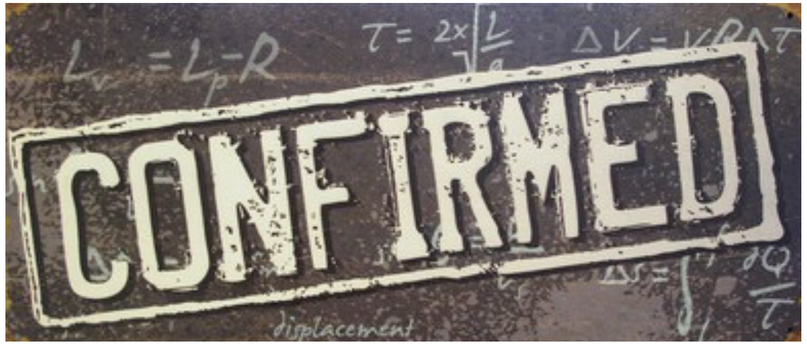
`std::regex` is too slow for production use



- difficult to use API
- very slow to compile
- very slow at runtime
- perf gotchas: regex c-tor, cmatch expensive

Myth #19

`std::regex` is too slow for production use



Use **CTRE** library instead:

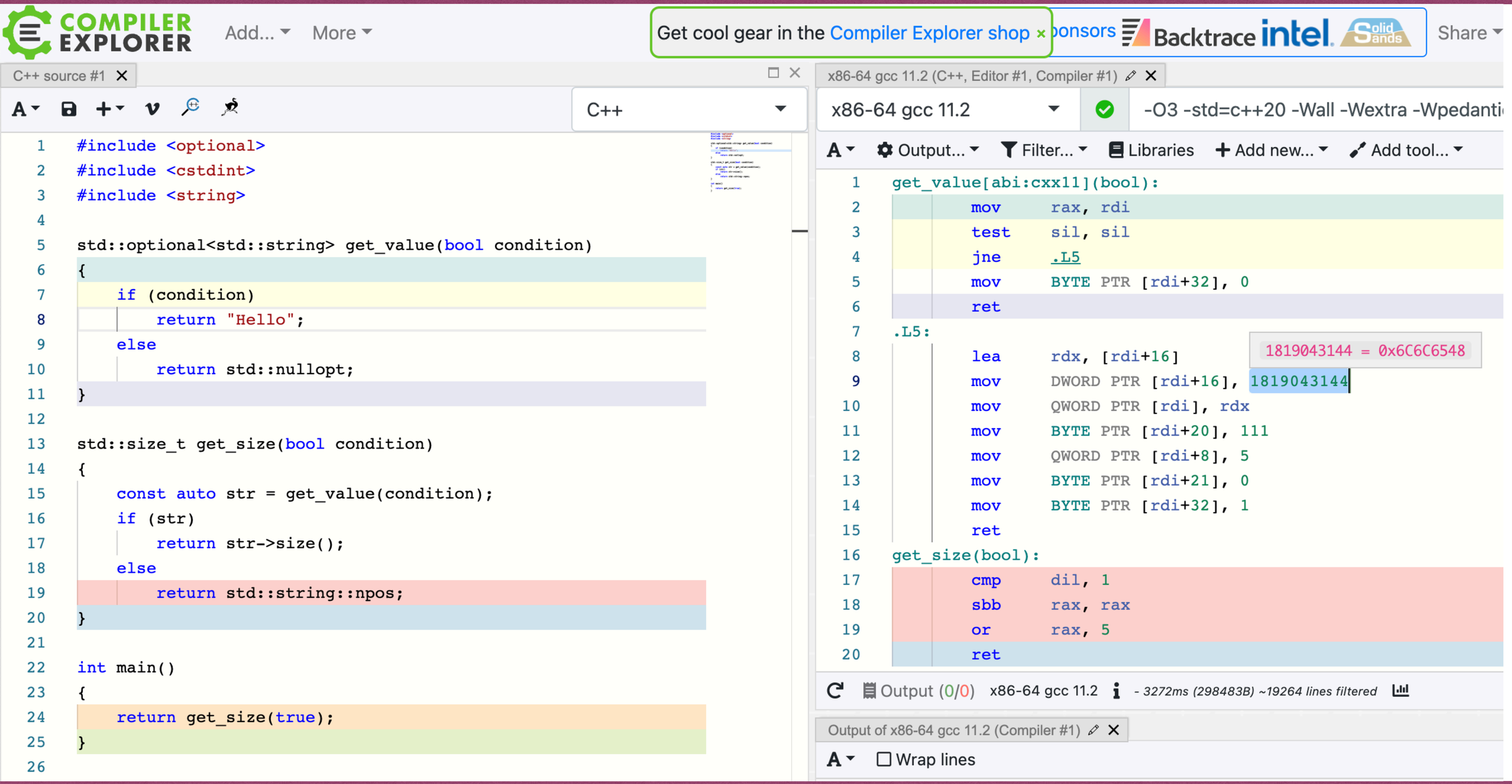
- very fast to compile
- much cleaner API
- supports `string_view`
- builds regular expressions automata at compile time
- github.com/hanickadot/compile-time-regular-expressions

Myth #24

`std::optional` inhibits optimizations and complicates APIs

Let's see...

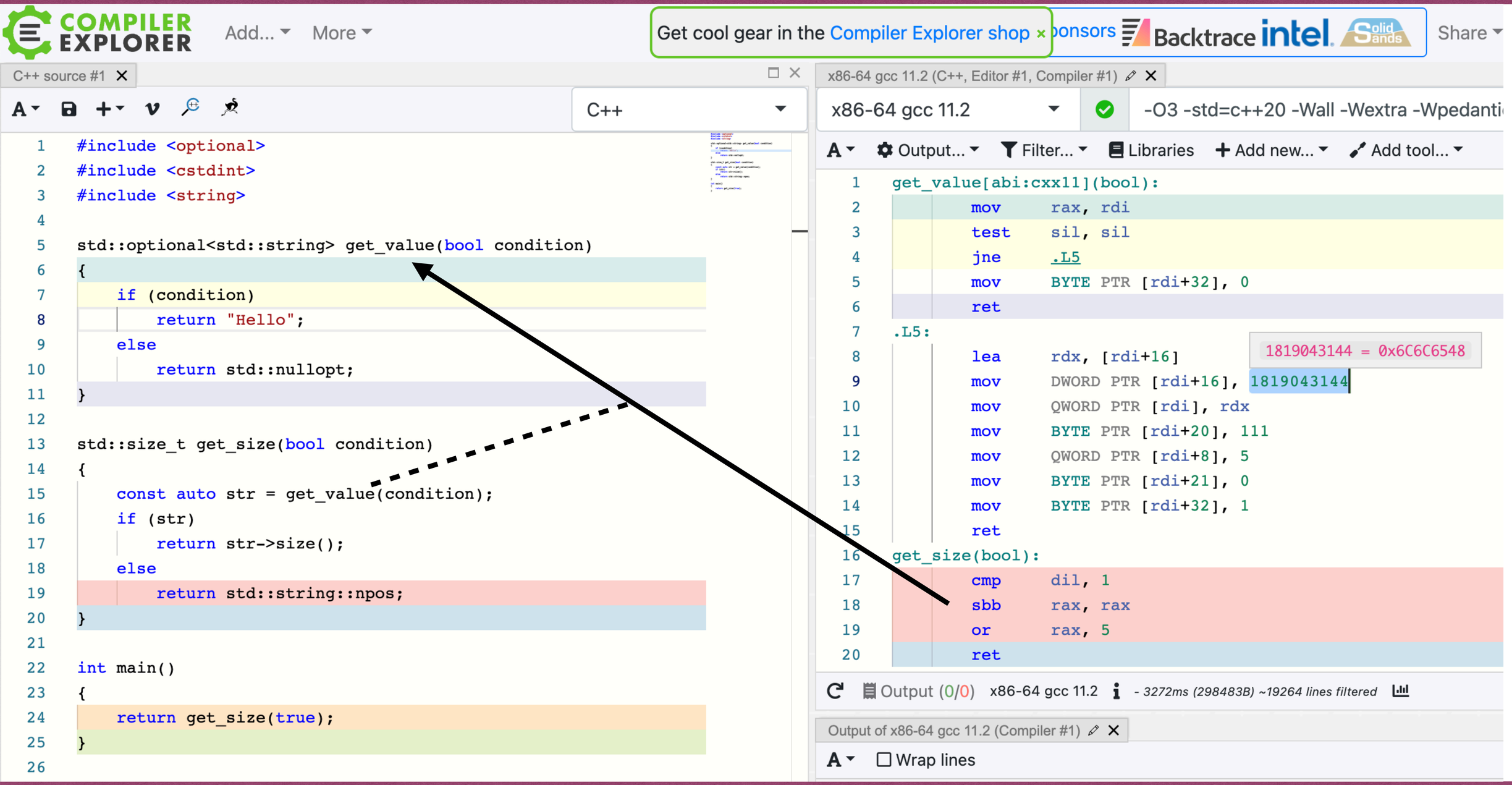
Myth #24




```cpp
#include <optional>
#include <cstdint>
#include <string>

std::optional<std::string> get_value(bool condition)
{
    if (condition)
        return "Hello";
    else
        return std::nullopt;
}

std::size_t get_size(bool condition)
{
    const auto str = get_value(condition);
    if (str)
        return str->size();
    else
        return std::string::npos;
}

int main()
{
    return get_size(true);
}
```

```asm
get_value[abi:cxx11](bool):
        mov     rax, rdi
        test    sil, sil
        jne     .L5
        mov     BYTE PTR [rdi+32], 0
        ret
.L5:
        lea     rdx, [rdi+16]
        mov     DWORD PTR [rdi+16], 1819043144
        mov     QWORD PTR [rdi], rdx
        mov     BYTE PTR [rdi+20], 111
        mov     QWORD PTR [rdi+8], 5
        mov     BYTE PTR [rdi+21], 0
        mov     BYTE PTR [rdi+32], 1
        ret
get_size(bool):
        cmp     dil, 1
        sbb     rax, rax
        or      rax, 5
        ret
```

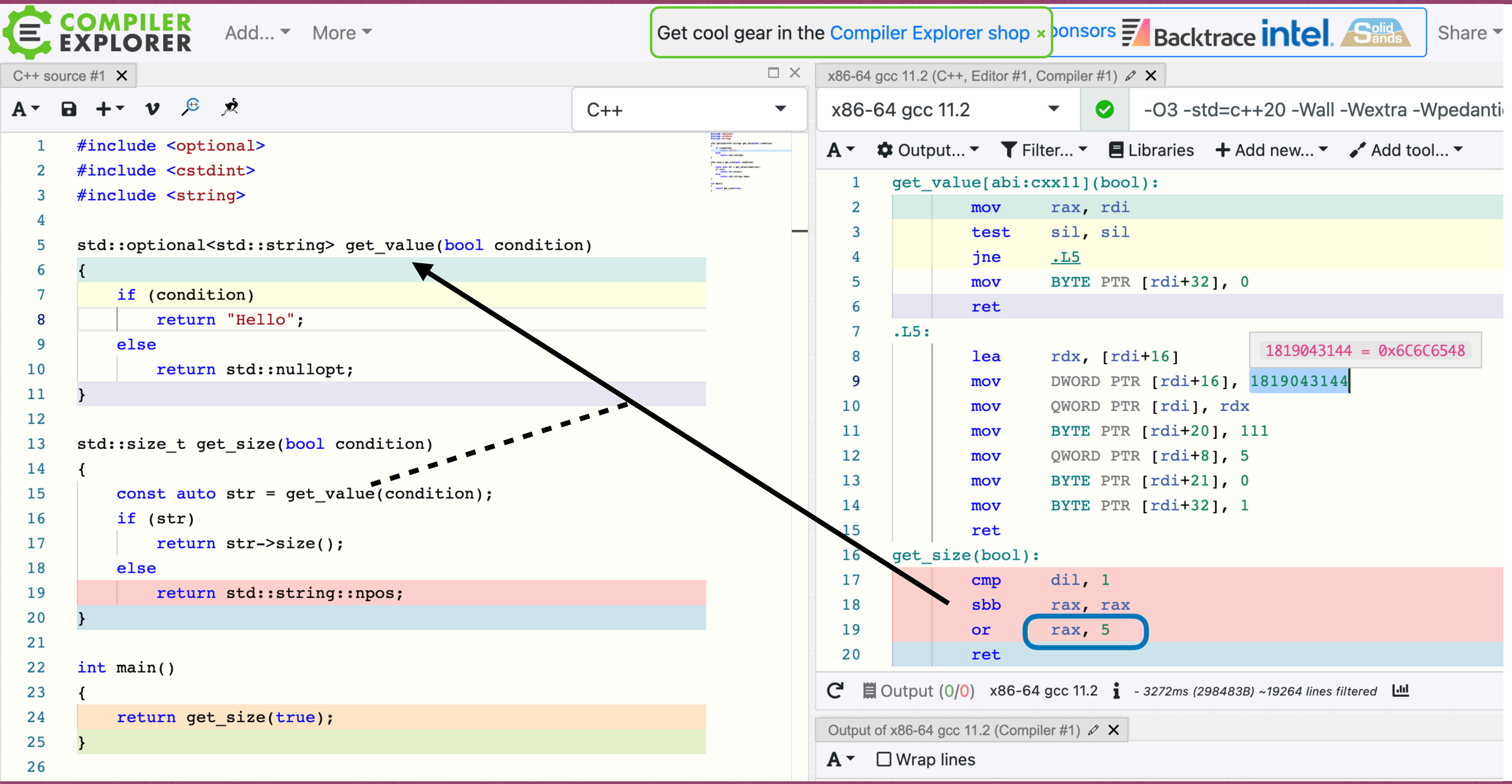
Myth #24



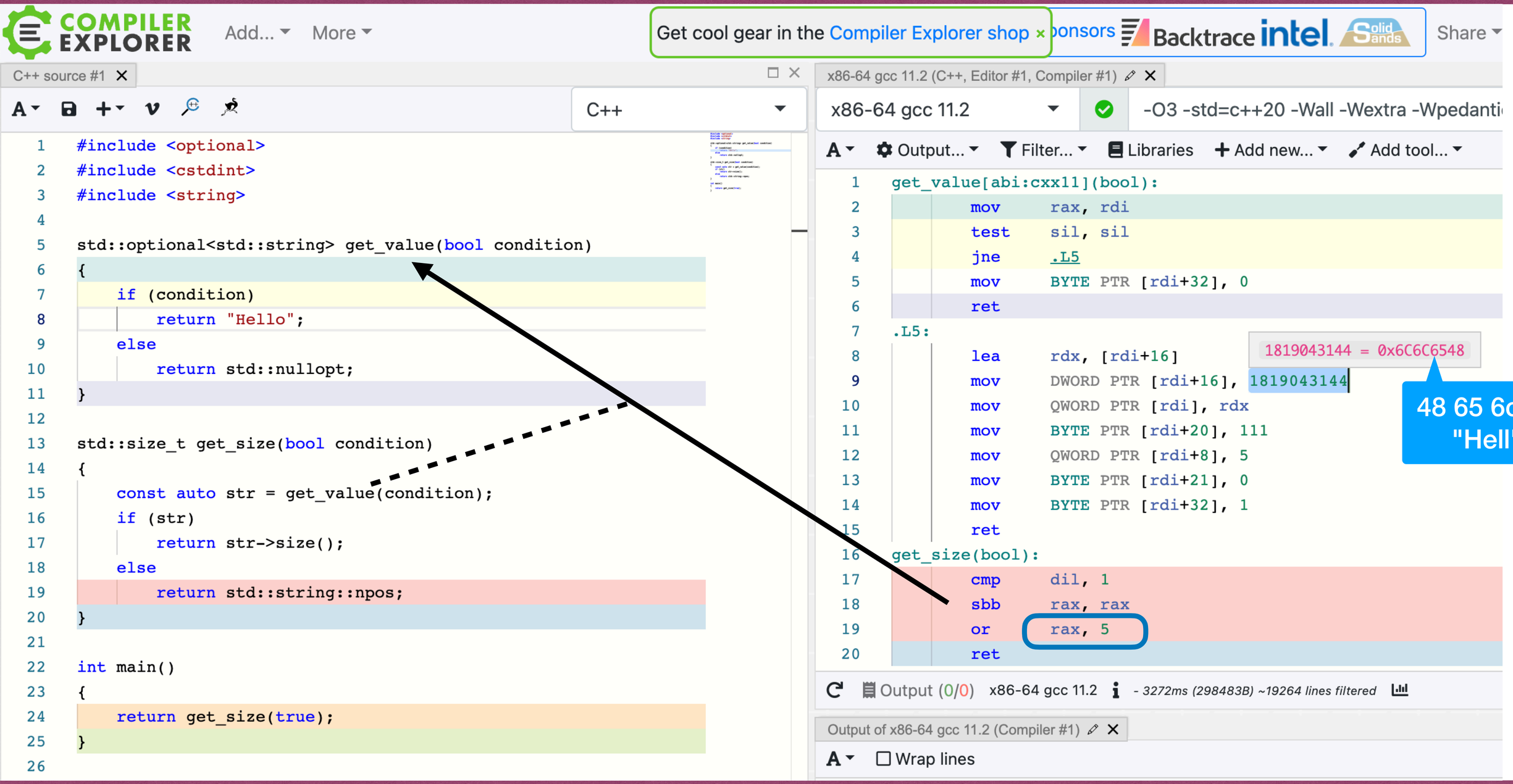

```cpp
#include <optional>
#include <cstdint>
#include <string>

std::optional<std::string> get_value(bool condition)
{
    if (condition)
        return "Hello";
    else
        return std::nullopt;
}

std::size_t get_size(bool condition)
{
    const auto str = get_value(condition);
    if (str)
        return str->size();
    else
        return std::string::npos;
}

int main()
{
    return get_size(true);
}
```

```asm
get_value[abi:cxx11](bool):
        mov     rax, rdi
        test    sil, sil
        jne     .L5
        mov     BYTE PTR [rdi+32], 0
        ret
.L5:
        lea     rdx, [rdi+16]
        mov     DWORD PTR [rdi+16], 1819043144
        mov     QWORD PTR [rdi], rdx
        mov     BYTE PTR [rdi+20], 111
        mov     QWORD PTR [rdi+8], 5
        mov     BYTE PTR [rdi+21], 0
        mov     BYTE PTR [rdi+32], 1
        ret
get_size(bool):
        cmp     dil, 1
        sbb     rax, rax
        or      rax, 5
        ret
```

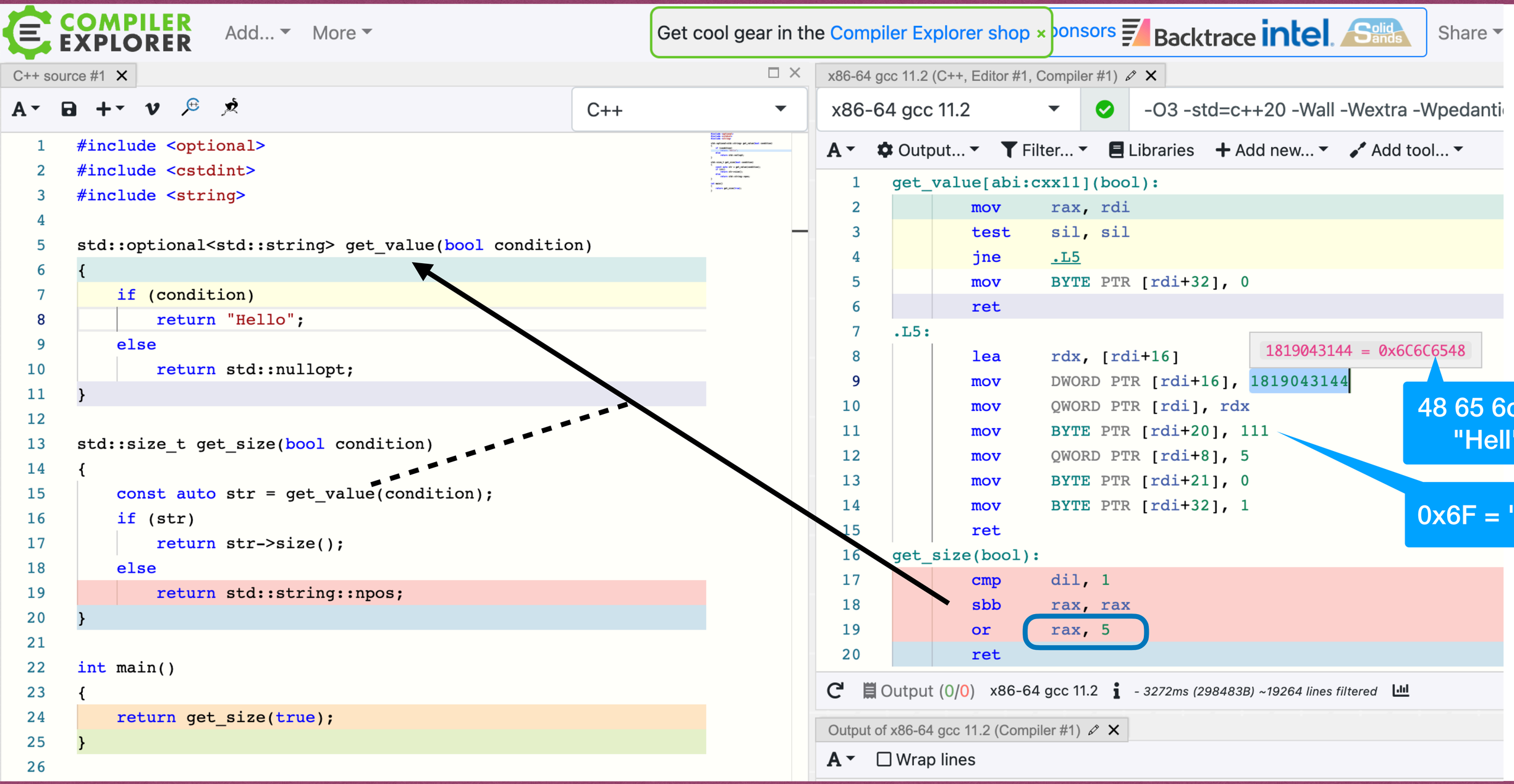
Myth #24


COMPILER EXPLORER
Add...
More
Get cool gear in the Compiler Explorer shop
Backtrace
intel
Share
C++ source #1
C++
x86-64 gcc 11.2 (C++, Editor #1, Compiler #1)
x86-64 gcc 11.2
-O3 -std=c++20 -Wall -Wextra -Wpedanti
Output...
Filter...
Libraries
Add new...
Add tool...
1 #include <optional>
2 #include <cstdint>
3 #include <string>
4
5 std::optional<std::string> get_value(bool condition)
6 {
7 if (condition)
8 return "Hello";
9 else
10 return std::nullopt;
11 }
12
13 std::size_t get_size(bool condition)
14 {
15 const auto str = get_value(condition);
16 if (str)
17 return str->size();
18 else
19 return std::string::npos;
20 }
21
22 int main()
23 {
24 return get_size(true);
25 }
26
get_value[abi:cxx11](bool):
mov rax, rdi
test sil, sil
jne .L5
mov BYTE PTR [rdi+32], 0
ret
.L5:
lea rdx, [rdi+16]
mov DWORD PTR [rdi+16], 1819043144
1819043144 = 0x6C6C6548
mov QWORD PTR [rdi], rdx
mov BYTE PTR [rdi+20], 111
mov QWORD PTR [rdi+8], 5
mov BYTE PTR [rdi+21], 0
mov BYTE PTR [rdi+32], 1
ret
get_size(bool):
cmp dil, 1
sbb rax, rax
or rax, 5
ret
Output (0/0) x86-64 gcc 11.2 - 3272ms (298483B) ~19264 lines filtered
Output of x86-64 gcc 11.2 (Compiler #1)
Wrap lines

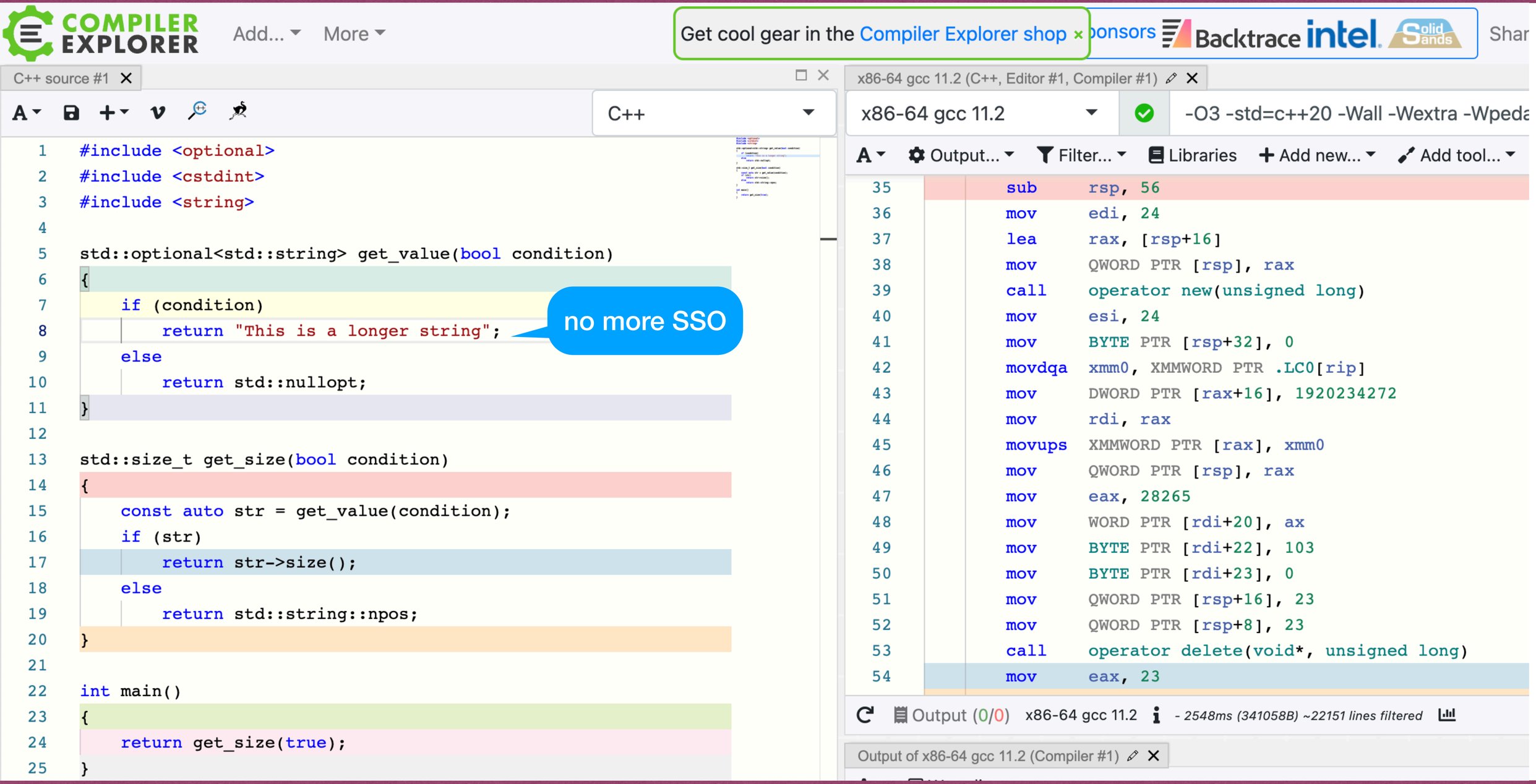
Myth #24


COMPILER EXPLORER
Add...
More
Get cool gear in the Compiler Explorer shop
Backtrace
intel
Share
C++ source #1
C++
x86-64 gcc 11.2 (C++, Editor #1, Compiler #1)
x86-64 gcc 11.2
-O3 -std=c++20 -Wall -Wextra -Wpedanti
Output...
Filter...
Libraries
Add new...
Add tool...
1 #include <optional>
2 #include <cstdint>
3 #include <string>
4
5 std::optional<std::string> get_value(bool condition)
6 {
7 if (condition)
8 return "Hello";
9 else
10 return std::nullopt;
11 }
12
13 std::size_t get_size(bool condition)
14 {
15 const auto str = get_value(condition);
16 if (str)
17 return str->size();
18 else
19 return std::string::npos;
20 }
21
22 int main()
23 {
24 return get_size(true);
25 }
26
get_value[abi:cxx11](bool):
mov rax, rdi
test sil, sil
jne .L5
mov BYTE PTR [rdi+32], 0
ret
.L5:
lea rdx, [rdi+16]
mov DWORD PTR [rdi+16], 1819043144
mov QWORD PTR [rdi], rdx
mov BYTE PTR [rdi+20], 111
mov QWORD PTR [rdi+8], 5
mov BYTE PTR [rdi+21], 0
mov BYTE PTR [rdi+32], 1
ret
get_size(bool):
cmp dil, 1
sbb rax, rax
or rax, 5
ret
1819043144 = 0x6C6C6548
48 65 6c 6c
"Hell"
Output (0/0) x86-64 gcc 11.2 - 3272ms (298483B) ~19264 lines filtered
Output of x86-64 gcc 11.2 (Compiler #1)
Wrap lines

Myth #24


COMPILER EXPLORER
Add...
More
Get cool gear in the Compiler Explorer shop
Backtrace
intel
Share
C++ source #1
C++
x86-64 gcc 11.2 (C++, Editor #1, Compiler #1)
x86-64 gcc 11.2
-O3 -std=c++20 -Wall -Wextra -Wpedanti
Output...
Filter...
Libraries
Add new...
Add tool...
1 #include <optional>
2 #include <cstdint>
3 #include <string>
4
5 std::optional<std::string> get_value(bool condition)
6 {
7 if (condition)
8 return "Hello";
9 else
10 return std::nullopt;
11 }
12
13 std::size_t get_size(bool condition)
14 {
15 const auto str = get_value(condition);
16 if (str)
17 return str->size();
18 else
19 return std::string::npos;
20 }
21
22 int main()
23 {
24 return get_size(true);
25 }
26
get_value[abi:cxx11](bool):
mov rax, rdi
test sil, sil
jne .L5
mov BYTE PTR [rdi+32], 0
ret
.L5:
lea rdx, [rdi+16]
mov DWORD PTR [rdi+16], 1819043144
mov QWORD PTR [rdi], rdx
mov BYTE PTR [rdi+20], 111
mov QWORD PTR [rdi+8], 5
mov BYTE PTR [rdi+21], 0
mov BYTE PTR [rdi+32], 1
ret
get_size(bool):
cmp dil, 1
sbb rax, rax
or rax, 5
ret
1819043144 = 0x6C6C6548
48 65 6c 6c
"Hell"
0x6F = 'o'
Output (0/0) x86-64 gcc 11.2 - 3272ms (298483B) ~19264 lines filtered
Output of x86-64 gcc 11.2 (Compiler #1)
Wrap lines

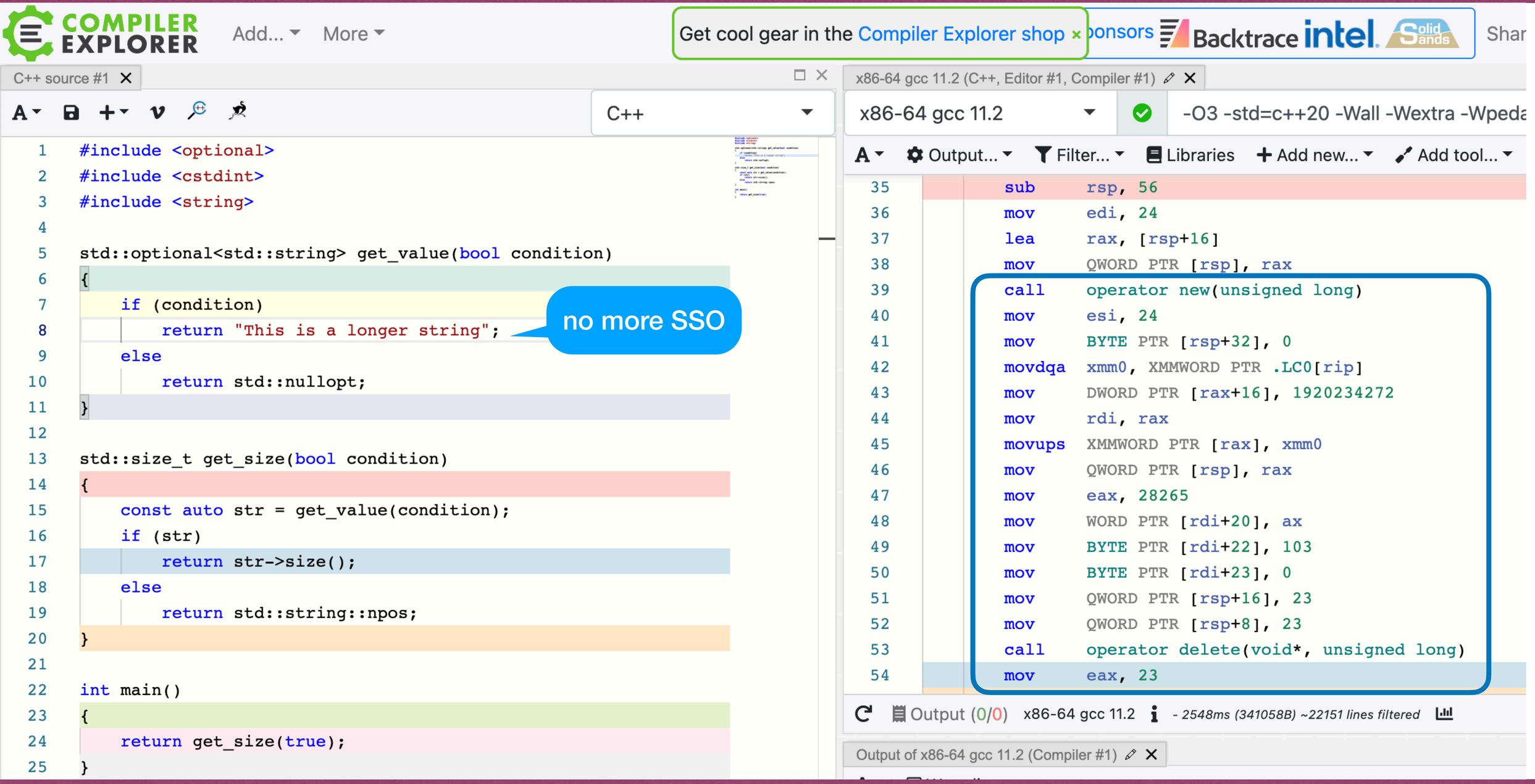
Myth #24




```
1  #include <optional>
2  #include <cstdint>
3  #include <string>
4
5  std::optional<std::string> get_value(bool condition)
6  {
7      if (condition)
8          return "This is a longer string";
9      else
10         return std::nullopt;
11 }
12
13 std::size_t get_size(bool condition)
14 {
15     const auto str = get_value(condition);
16     if (str)
17         return str->size();
18     else
19         return std::string::npos;
20 }
21
22 int main()
23 {
24     return get_size(true);
25 }
```

```
35      sub     rsp, 56
36      mov     edi, 24
37      lea     rax, [rsp+16]
38      mov     QWORD PTR [rsp], rax
39      call    operator new(unsigned long)
40      mov     esi, 24
41      mov     BYTE PTR [rsp+32], 0
42      movdqa  xmm0, XMMWORD PTR .LC0[rip]
43      mov     DWORD PTR [rax+16], 1920234272
44      mov     rdi, rax
45      movups  XMMWORD PTR [rax], xmm0
46      mov     QWORD PTR [rsp], rax
47      mov     eax, 28265
48      mov     WORD PTR [rdi+20], ax
49      mov     BYTE PTR [rdi+22], 103
50      mov     BYTE PTR [rdi+23], 0
51      mov     QWORD PTR [rsp+16], 23
52      mov     QWORD PTR [rsp+8], 23
53      call    operator delete(void*, unsigned long)
54      mov     eax, 23
```

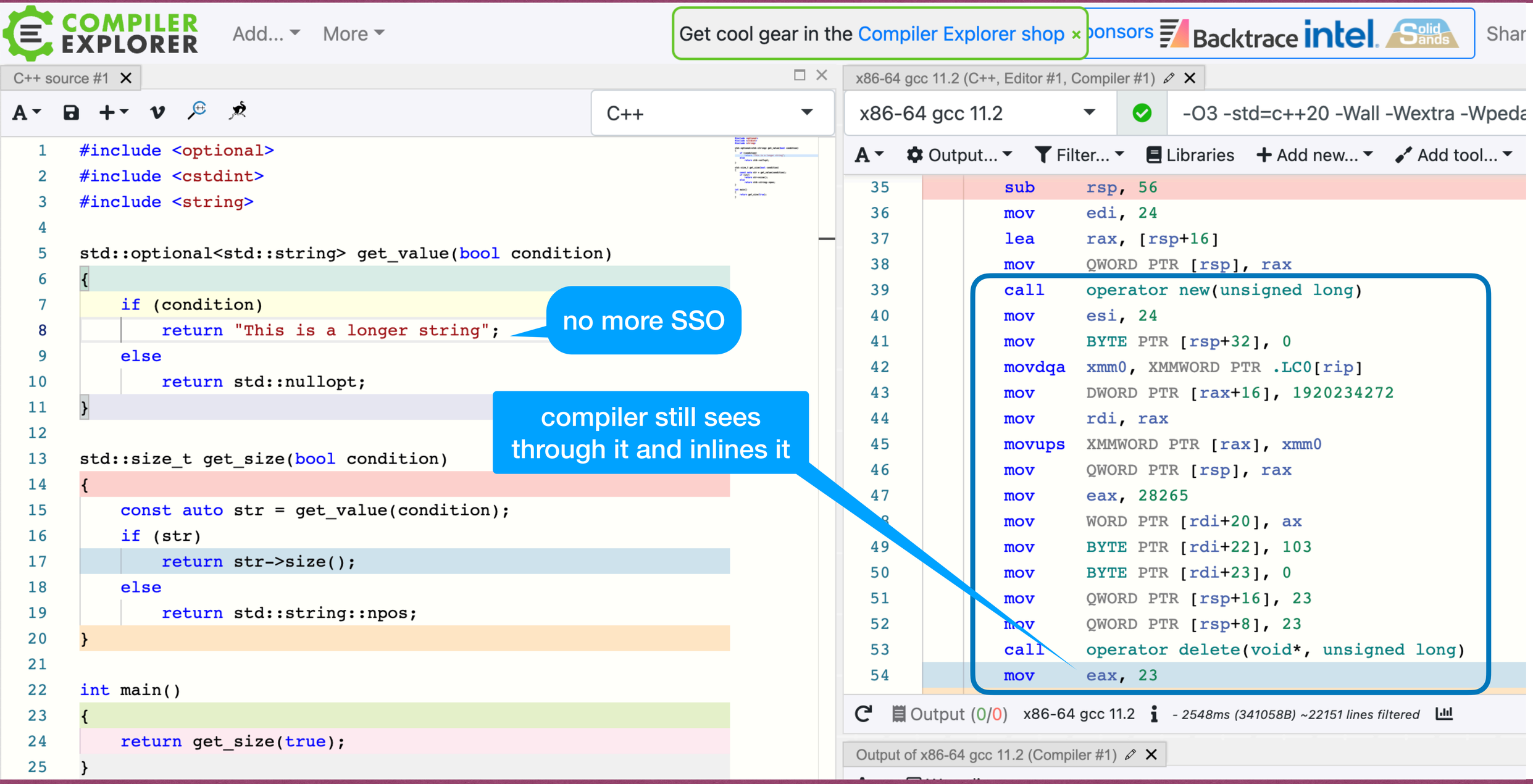
Myth #24




```cpp
#include <optional>
#include <cstdint>
#include <string>

std::optional<std::string> get_value(bool condition)
{
    if (condition)
        return "This is a longer string";
    else
        return std::nullopt;
}

std::size_t get_size(bool condition)
{
    const auto str = get_value(condition);
    if (str)
        return str->size();
    else
        return std::string::npos;
}

int main()
{
    return get_size(true);
}
```

```
sub     rsp, 56
mov     edi, 24
lea     rax, [rsp+16]
mov     QWORD PTR [rsp], rax
call    operator new(unsigned long)
mov     esi, 24
mov     BYTE PTR [rsp+32], 0
movdqa  xmm0, XMMWORD PTR .LC0[rip]
mov     DWORD PTR [rax+16], 1920234272
mov     rdi, rax
movups  XMMWORD PTR [rax], xmm0
mov     QWORD PTR [rsp], rax
mov     eax, 28265
mov     WORD PTR [rdi+20], ax
mov     BYTE PTR [rdi+22], 103
mov     BYTE PTR [rdi+23], 0
mov     QWORD PTR [rsp+16], 23
mov     QWORD PTR [rsp+8], 23
call    operator delete(void*, unsigned long)
mov     eax, 23
```

Myth #24

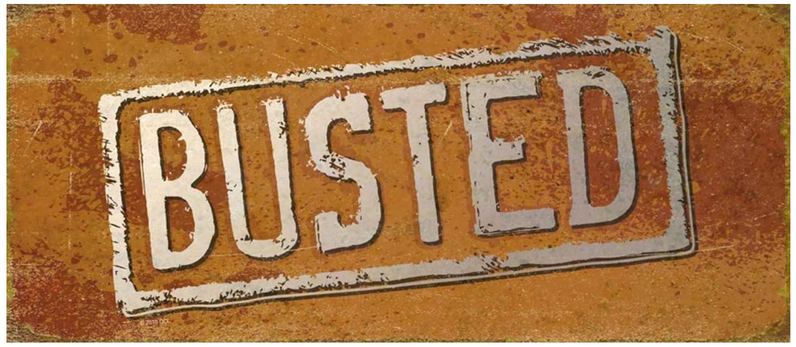



```
1  #include <optional>
2  #include <cstdint>
3  #include <string>
4
5  std::optional<std::string> get_value(bool condition)
6  {
7      if (condition)
8          return "This is a longer string";
9      else
10         return std::nullopt;
11 }
12
13 std::size_t get_size(bool condition)
14 {
15     const auto str = get_value(condition);
16     if (str)
17         return str->size();
18     else
19         return std::string::npos;
20 }
21
22 int main()
23 {
24     return get_size(true);
25 }
```

```
35      sub     rsp, 56
36      mov     edi, 24
37      lea     rax, [rsp+16]
38      mov     QWORD PTR [rsp], rax
39      call    operator new(unsigned long)
40      mov     esi, 24
41      mov     BYTE PTR [rsp+32], 0
42      movdqa  xmm0, XMMWORD PTR .LC0[rip]
43      mov     DWORD PTR [rax+16], 1920234272
44      mov     rdi, rax
45      movups  XMMWORD PTR [rax], xmm0
46      mov     QWORD PTR [rsp], rax
47      mov     eax, 28265
        mov     WORD PTR [rdi+20], ax
49      mov     BYTE PTR [rdi+22], 103
50      mov     BYTE PTR [rdi+23], 0
51      mov     QWORD PTR [rsp+16], 23
52      mov     QWORD PTR [rsp+8], 23
53      call    operator delete(void*, unsigned long)
54      mov     eax, 23
```

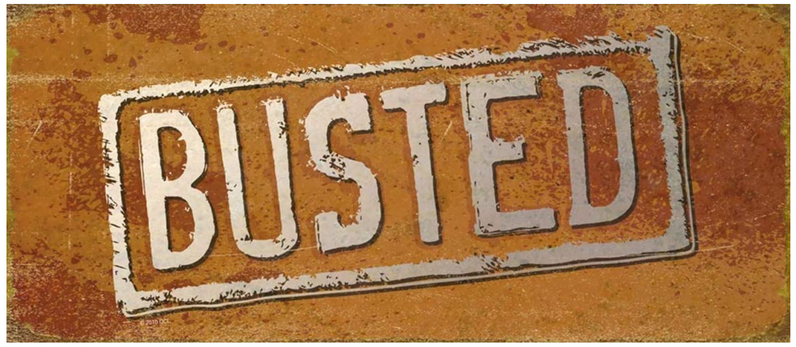
Myth #24

`std::optional` inhibits optimizations

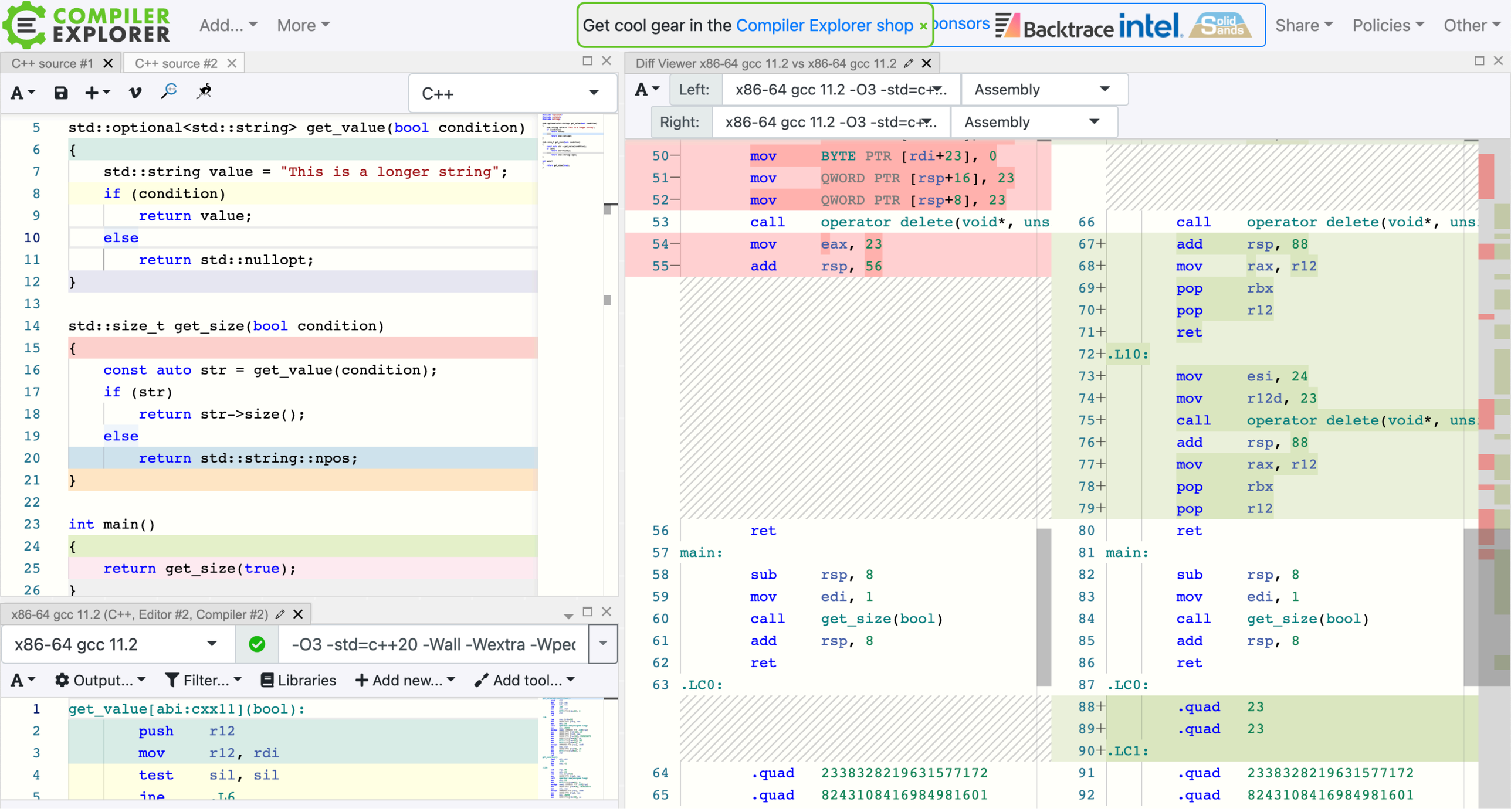


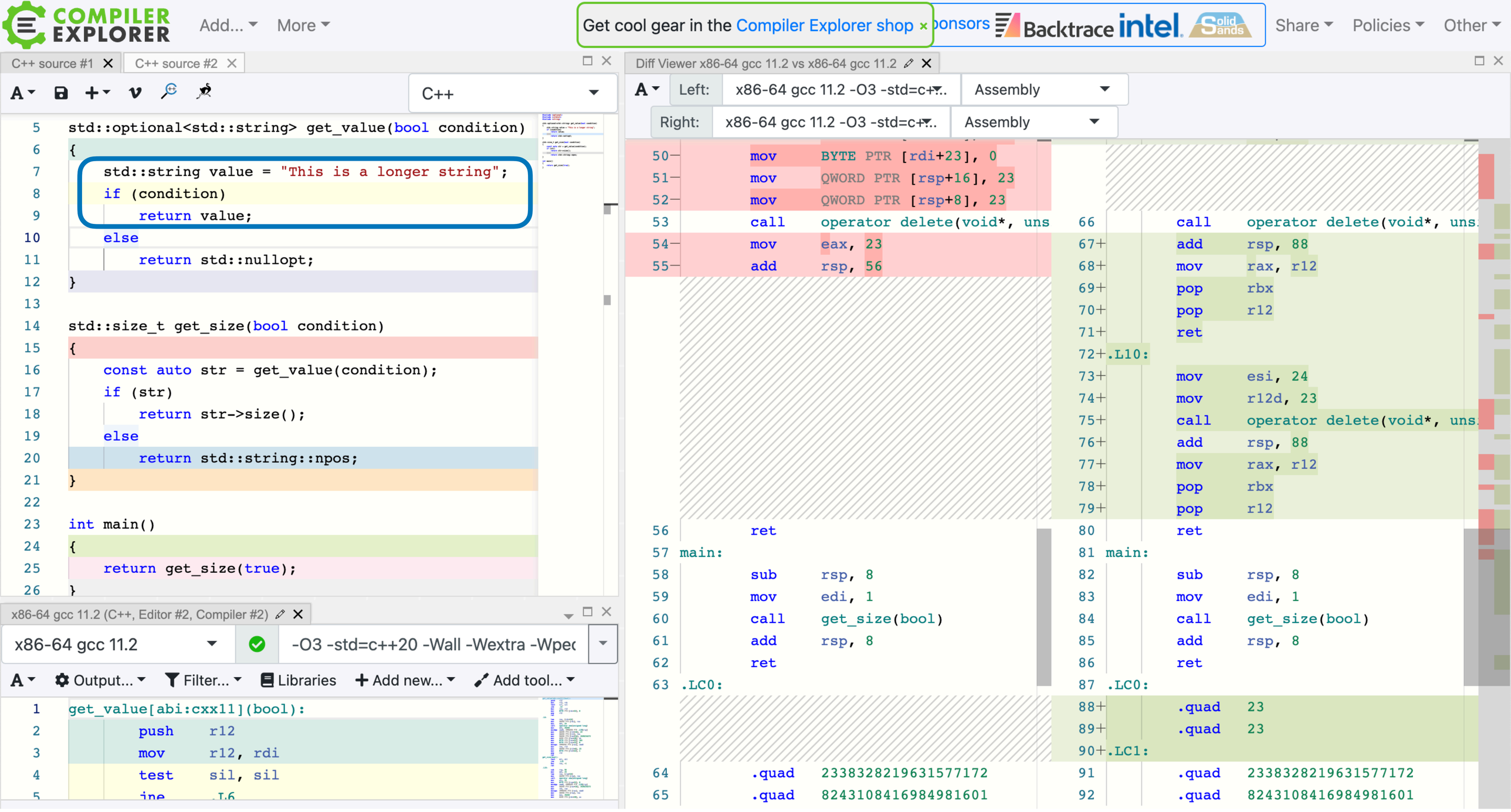
Myth #24

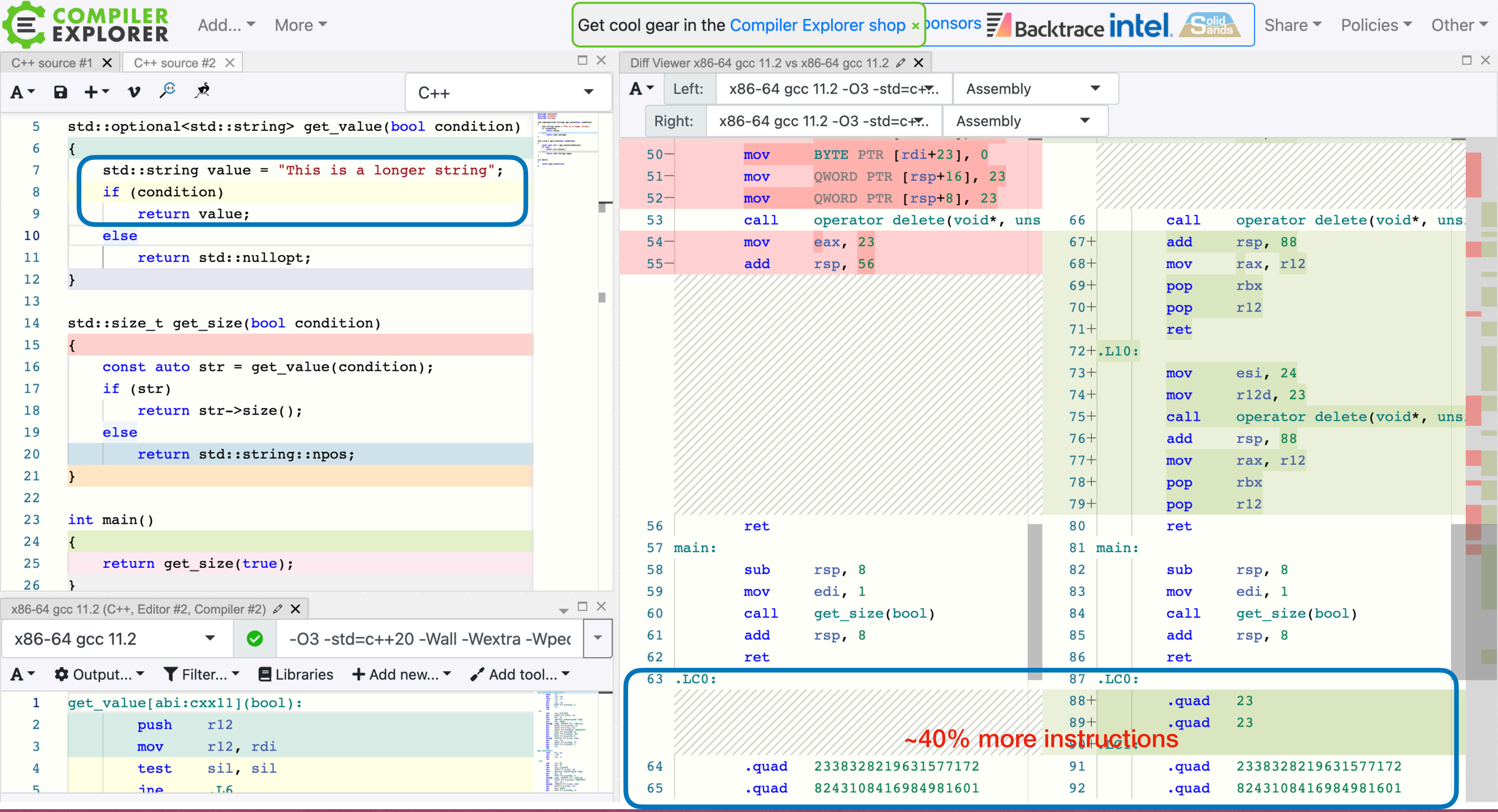
`std::optional` inhibits optimizations

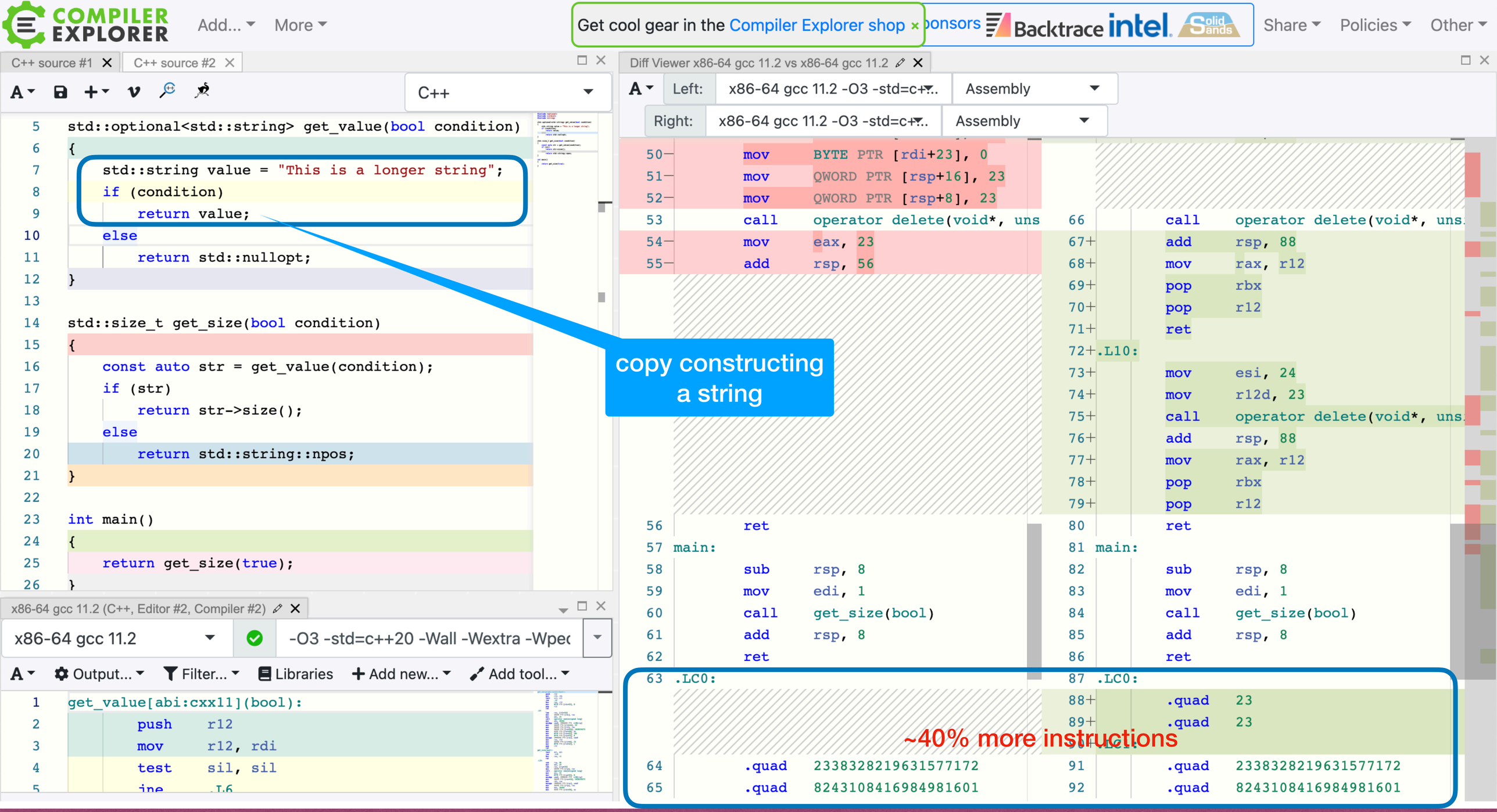


However...


COMPILER EXPLORER
Add...
More
Get cool gear in the Compiler Explorer shop
Backtrace
intel
Share
Policies
Other
C++ source #1
C++ source #2
C++
std::optional<std::string> get_value(bool condition)
{
std::string value = "This is a longer string";
if (condition)
return value;
else
return std::nullopt;
}
std::size_t get_size(bool condition)
{
const auto str = get_value(condition);
if (str)
return str->size();
else
return std::string::npos;
}
int main()
{
return get_size(true);
}
x86-64 gcc 11.2 (C++, Editor #2, Compiler #2)
x86-64 gcc 11.2
-O3 -std=c++20 -Wall -Wextra -Wpe
Output...
Filter...
Libraries
Add new...
Add tool...
get_value[abi:cxx11](bool):
push r12
mov r12, rdi
test sil, sil
Diff Viewer x86-64 gcc 11.2 vs x86-64 gcc 11.2
Left:
x86-64 gcc 11.2 -O3 -std=c++..
Assembly
Right:
x86-64 gcc 11.2 -O3 -std=c++..
Assembly
mov BYTE PTR [rdi+23], 0
mov QWORD PTR [rsp+16], 23
mov QWORD PTR [rsp+8], 23
call operator delete(void*, uns
mov eax, 23
add rsp, 56
ret
main:
sub rsp, 8
mov edi, 1
call get_size(bool)
add rsp, 8
ret
.LC0:
.quad 2338328219631577172
.quad 8243108416984981601
call operator delete(void*, uns
add rsp, 88
mov rax, r12
pop rbx
pop r12
ret
.L10:
mov esi, 24
mov r12d, 23
call operator delete(void*, uns
add rsp, 88
mov rax, r12
pop rbx
pop r12
ret
main:
sub rsp, 8
mov edi, 1
call get_size(bool)
add rsp, 8
ret
.LC0:
.quad 23
.quad 23
.LC1:
.quad 2338328219631577172
.quad 8243108416984981601


COMPILER EXPLORER
Add...
More
Get cool gear in the Compiler Explorer shop
Backtrace
intel
Share
Policies
Other
C++ source #1
C++ source #2
C++
5 std::optional<std::string> get_value(bool condition)
6 {
7 std::string value = "This is a longer string";
8 if (condition)
9 return value;
10 else
11 return std::nullopt;
12 }
13
14 std::size_t get_size(bool condition)
15 {
16 const auto str = get_value(condition);
17 if (str)
18 return str->size();
19 else
20 return std::string::npos;
21 }
22
23 int main()
24 {
25 return get_size(true);
26 }
x86-64 gcc 11.2 (C++, Editor #2, Compiler #2)
x86-64 gcc 11.2
-O3 -std=c++20 -Wall -Wextra -Wpe
Output...
Filter...
Libraries
Add new...
Add tool...
1 get_value[abi:cxx11](bool):
2 push r12
3 mov r12, rdi
4 test sil, sil
Diff Viewer x86-64 gcc 11.2 vs x86-64 gcc 11.2
Left: x86-64 gcc 11.2 -O3 -std=c++..
Assembly
Right: x86-64 gcc 11.2 -O3 -std=c++..
Assembly
50 mov BYTE PTR [rdi+23], 0
51 mov QWORD PTR [rsp+16], 23
52 mov QWORD PTR [rsp+8], 23
53 call operator delete(void*, uns
54 mov eax, 23
55 add rsp, 56
56 ret
57 main:
58 sub rsp, 8
59 mov edi, 1
60 call get_size(bool)
61 add rsp, 8
62 ret
63 .LC0:
64 .quad 2338328219631577172
65 .quad 8243108416984981601
66 call operator delete(void*, uns
67+ add rsp, 88
68+ mov rax, r12
69+ pop rbx
70+ pop r12
71+ ret
72+.L10:
73+ mov esi, 24
74+ mov r12d, 23
75+ call operator delete(void*, uns
76+ add rsp, 88
77+ mov rax, r12
78+ pop rbx
79+ pop r12
80 ret
81 main:
82 sub rsp, 8
83 mov edi, 1
84 call get_size(bool)
85 add rsp, 8
86 ret
87 .LC0:
88+ .quad 23
89+ .quad 23
90+.LC1:
91 .quad 2338328219631577172
92 .quad 8243108416984981601


COMPILER EXPLORER
Add...
More
Get cool gear in the Compiler Explorer shop
Backtrace
intel
Share
Policies
Other
C++ source #1
C++ source #2
C++
std::optional<std::string> get_value(bool condition)
{
std::string value = "This is a longer string";
if (condition)
return value;
else
return std::nullopt;
}
std::size_t get_size(bool condition)
{
const auto str = get_value(condition);
if (str)
return str->size();
else
return std::string::npos;
}
int main()
{
return get_size(true);
}
x86-64 gcc 11.2 (C++, Editor #2, Compiler #2)
x86-64 gcc 11.2
-O3 -std=c++20 -Wall -Wextra -Wpe
Output...
Filter...
Libraries
Add new...
Add tool...
get_value[abi:cxx11](bool):
push r12
mov r12, rdi
test sil, sil
Diff Viewer x86-64 gcc 11.2 vs x86-64 gcc 11.2
Left:
x86-64 gcc 11.2 -O3 -std=c++..
Assembly
Right:
x86-64 gcc 11.2 -O3 -std=c++..
Assembly
mov BYTE PTR [rdi+23], 0
mov QWORD PTR [rsp+16], 23
mov QWORD PTR [rsp+8], 23
call operator delete(void*, uns
mov eax, 23
add rsp, 56
ret
main:
sub rsp, 8
mov edi, 1
call get_size(bool)
add rsp, 8
ret
.LC0:
.quad 2338328219631577172
.quad 824310841698498160
call operator delete(void*, uns
add rsp, 88
mov rax, r12
pop rbx
pop r12
ret
.L10:
mov esi, 24
mov r12d, 23
call operator delete(void*, uns
add rsp, 88
mov rax, r12
pop rbx
pop r12
ret
main:
sub rsp, 8
mov edi, 1
call get_size(bool)
add rsp, 8
ret
.LC0:
.quad 23
.quad 23
.quad 2338328219631577172
.quad 824310841698498160
~40% more instructions


COMPILER EXPLORER
Add...
More
Get cool gear in the Compiler Explorer shop
Share
Policies
Other
C++ source #1
C++ source #2
C++
std::optional<std::string> get_value(bool condition)
{
std::string value = "This is a longer string";
if (condition)
return value;
else
return std::nullopt;
}
std::size_t get_size(bool condition)
{
const auto str = get_value(condition);
if (str)
return str->size();
else
return std::string::npos;
}
int main()
{
return get_size(true);
}
copy constructing a string
x86-64 gcc 11.2 (C++, Editor #2, Compiler #2)
x86-64 gcc 11.2
-O3 -std=c++20 -Wall -Wextra -Wpe
Output...
Filter...
Libraries
Add new...
Add tool...
get_value[abi:cxx11](bool):
push r12
mov r12, rdi
test sil, sil
Diff Viewer x86-64 gcc 11.2 vs x86-64 gcc 11.2
Left: x86-64 gcc 11.2 -O3 -std=c++..
Assembly
Right: x86-64 gcc 11.2 -O3 -std=c++..
Assembly
mov BYTE PTR [rdi+23], 0
mov QWORD PTR [rsp+16], 23
mov QWORD PTR [rsp+8], 23
call operator delete(void*, uns
mov eax, 23
add rsp, 56
ret
main:
sub rsp, 8
mov edi, 1
call get_size(bool)
add rsp, 8
ret
.LC0:
.quad 2338328219631577172
.quad 824310841698498160
call operator delete(void*, uns
add rsp, 88
mov rax, r12
pop rbx
pop r12
ret
.L10:
mov esi, 24
mov r12d, 23
call operator delete(void*, uns
add rsp, 88
mov rax, r12
pop rbx
pop r12
ret
main:
sub rsp, 8
mov edi, 1
call get_size(bool)
add rsp, 8
ret
.LC0:
.quad 23
.quad 23
.quad 2338328219631577172
.quad 824310841698498160
~40% more instructions

Myth #24

```
template <class U = T>
constexpr optional(U && value);
```

Constructs an optional object that contains a value, initialized ***as if*** direct-initializing (but not direct-list-initializing) an object of type T with `std::forward<U>(value)`

- this constructor does not participate in overload resolution
  unless `std::is_constructible_v<T, U&&>` is *true* and `std::remove_cvref_t<U>` is neither `std::in_place_t` nor `std::optional<T>`

- this constructor is explicit iff `std::is_convertible_v<U&&, T>` is *false*

Myth #24

`std::optional` complicates APIs

Myth #24

`std::optional` complicates APIs

- don't look inside the 📦 box
- don't use optional for error handling
- when in doubt, draw inspiration from other APIs:

  Haskell (`Maybe`) or Rust (`Option<T>`)

Myth #24

`std::optional` complicates APIs

- don't look inside the 📦 box
- don't use optional for error handling
- when in doubt, draw inspiration from other APIs:

  Haskell (`Maybe`) or Rust (`Option<T>`)

Myth #24

`std::optional` complicates APIs

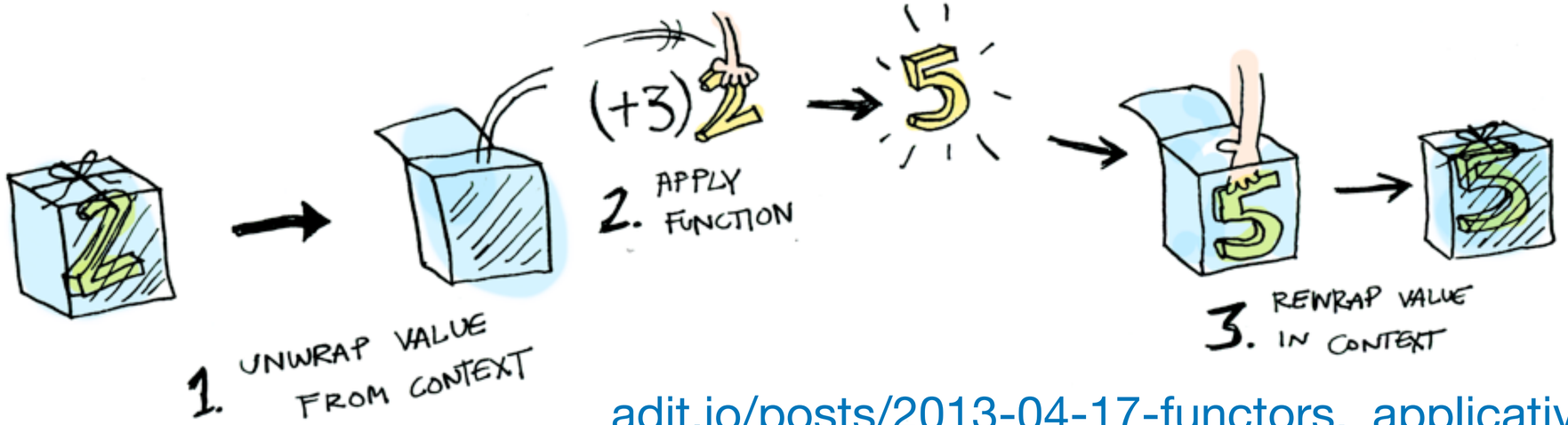
- don't look inside the 📦 box
- don't use optional for error handling
- when in doubt, draw inspiration from other APIs:

  Haskell (`Maybe`) or Rust (`Option<T>`)

adit.io/posts/2013-04-17-functors,_applicatives,_and_monads_in_pictures

Myth #24

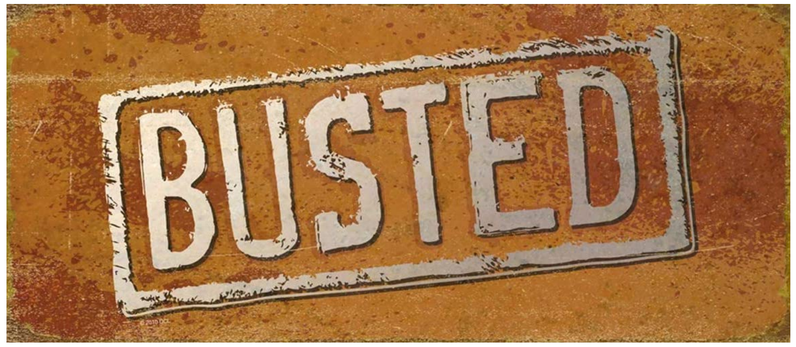
`std::optional` complicates APIs



adit.io/posts/2013-04-17-functors,_applicatives,_and_monads_in_pictures

Myth #24

`std::optional` complicates APIs



Myth #31

`std::move()` moves ?

Myth #31

`std::move()` moves ?

```
void echo(const std::string & first, const std::string & second)
{
  fmt::print("'{}','{}'", first, second);
}

int main()
{
  std::string greeting{"Hello from a long string"};
  echo(greeting, greeting);
}
```

Myth #31

`std::move()` moves ?

```
void echo(const std::string & first, const std::string & second)
{
  fmt::print("'{}','{}'", first, second);
}

int main()
{
  std::string greeting{"Hello from a long string"};
  echo(greeting, greeting);
}
```

```
'Hello from a long string','Hello from a long string'
```

Myth #31

`std::move()` moves ?

```
void echo(const std::string & first, const std::string & second)
{
  fmt::print("'{}','{}'", first, second);
}

int main()
{
  std::string greeting{"Hello from a long string"};
  echo(std::move(greeting), greeting);
}
```

Myth #31

`std::move()` moves ?

```
void echo(const std::string & first, const std::string & second)
{
  fmt::print("'{}','{}'", first, second);
}

int main()
{
  std::string greeting{"Hello from a long string"};
  echo(std::move(greeting), greeting);
}
```

```
'Hello from a long string','Hello from a long string'
```

Myth #31

`std::move()` moves ?

```
void echo(const std::string & first, const std::string & second)
{
  fmt::print("'{}','{}'", first, second);
}

int main()
{
  std::string greeting{"Hello from a long string"};
  echo(std::move(greeting), std::move(greeting));
}
```

Myth #31

`std::move()` moves ?

```
void echo(const std::string & first, const std::string & second)
{
  fmt::print("'{}','{}'", first, second);
}

int main()
{
  std::string greeting{"Hello from a long string"};
  echo(std::move(greeting), std::move(greeting));
}
```

```
'Hello from a long string','Hello from a long string'
```

Myth #31

`std::move()` moves ?

```
void echo(const std::string & first, const std::string & second)
{
  fmt::print("'{}','{}'", first, second);
}

int main()
{
  std::string greeting{"Hello from a long string"};
  echo(std::move(greeting), std::move(greeting));
}
```

string && => const string &

```
'Hello from a long string','Hello from a long string'
```

Myth #31

`std::move()` moves ?

```
void echo(const std::string first, const std::string second)
{
  fmt::print("'{}','{}'", first, second);
}

int main()
{
  std::string greeting{"Hello from a long string"};
  echo(std::move(greeting), std::move(greeting));
}
```

Myth #31

`std::move()` moves ?

```
void echo(const std::string first, const std::string second)
{
  fmt::print("'{}','{}'", first, second);
}

int main()
{
  std::string greeting{"Hello from a long string"};
  echo(std::move(greeting), std::move(greeting));
}
```

```
'Hello from a long string',''
```

Myth #31

`std::move()` moves ?

```
void echo(const std::string first, const std::string second)
{
  fmt::print("'{}','{}'", first, second);
}

int main()
{
  std::string greeting{"Hello from a long string"};
  echo(std::move(greeting), std::move(greeting));
}
```

`string(std::move(greeting))`

```
'Hello from a long string',''
```

Myth #31

`std::move()` moves ?

```
void echo(const std::string first, const std::string second)
{
  fmt::print("'{}','{}'", first, second);
}

int main()
{
  std::string greeting{"Hello from a long string"};
  echo(std::move(greeting), std::move(greeting));
}
```

`string(std::move(greeting))`

clang

```
'Hello from a long string',''
```

Myth #31

`std::move()` moves ?

```
void echo(const std::string first, const std::string second)
{
  fmt::print("'{}','{}'", first, second);
}

int main()
{
  std::string greeting{"Hello from a long string"};
  echo(std::move(greeting), std::move(greeting));
}
```

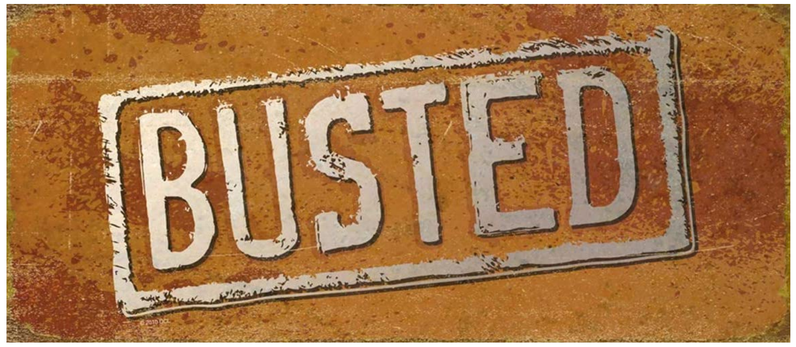
`string(std::move(greeting))`

gcc

```
'','Hello from a long string'
```

Myth #31

`std::move()` moves ?



Myth #36

Always pass input arguments
by const reference

```
void echo(const std::string & first, const std::string & second);
```

Myth #36

```
class Widget
{
  std::string id;

public:

  Widget(const std::string & new_id)
  : id(new_id) {}

  Widget(std::string && new_id)
  : id(std::move(new_id)) {}

};
```

Myth #36

```
class Widget
{
  std::string id;
  std::string name;

public:

  Widget(const std::string & new_id, const std::string & new_name)
  : id(new_id), name(new_name) {}

  Widget(std::string && new_id, std::string && new_name)
  : id(std::move(new_id)), name(std::move(new_name)) {}

  Widget(const std::string & new_id, std::string && new_name)
  : id(new_id), name(std::move(new_name)) {}

  Widget(std::string && new_id, const std::string & new_name)
  : id(std::move(new_id)), name(new_name) {}

};
```

Myth #36

```
class Widget
{
  std::string id;
  std::string name;

public:

  Widget(std::string new_id, std::string new_name)
  : id(std::move(new_id)), name(std::move(new_name)) {}

};
```

when we take ownership (sink)

by value

Myth #36

```
class Widget
{
  std::string id;
  std::string name;

public:

  void set_name(std::string new_name)
  {
    name = std::move(new_name);
  }

};
```

by value

when we take ownership (sink)

Myth #36

```
class Widget
{
  std::string id;
  std::string name;

public:

  void set_name(std::string new_name)
  {
    name = std::move(new_name);
  }
};

Widget w;
w.set_name("Hello from a long string");
```

- create the string with the literal value
- move assignment into data member

Myth #36

```
class Widget
{
  std::string id;
  std::string name;

public:

  void set_name(std::string new_name)
  {
    name = std::move(new_name);
  }
};

Widget w;
std::string name{"Hello from a long string"};
w.set_name(name);
```

- create the string with the literal value
- make a copy of the string
- move assignment into data member

Myth #36

```
class Widget
{
  std::string id;
  std::string name;

public:

  void set_name(std::string new_name)
  {
    name = std::move(new_name);
  }
};

Widget w;
std::string name{"Hello from a long string"};
w.set_name(std::move(name));
```

- create the string with the literal value
- move construct the string
- move assignment into data member

Myth #36

```
class Widget
{
  std::string name;

public:

  void set_name(const std::string & new_name)
  {
    name = new_name;
  }
  void set_name(std::string && new_name)
  {
    name = std::move(new_name);
  }
};

Widget w;
std::string name{"Hello from a long string"};
w.set_name(name);
```

- create the string with the literal value
- make a copy of the string

Myth #36

```
class Widget
{
  std::string name;

public:

  void set_name(const std::string & new_name)
  {
    name = new_name;
  }
  void set_name(std::string && new_name)
  {
    name = std::move(new_name);
  }
};

Widget w;
std::string name{"Hello from a long string"};
w.set_name(name);
```

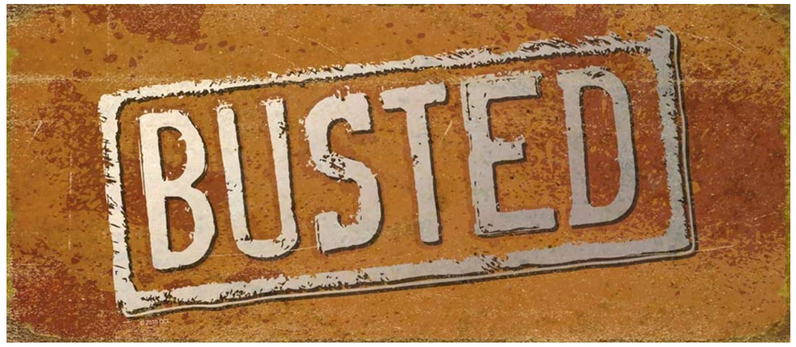
Technically, more efficient

(one less move operation)

- create the string with the literal value
- make a copy of the string

Myth #36

Always pass input arguments
by const reference



Myth #36

Always pass input arguments
by const reference

There's even a clang-tidy *modernizer* check to perform this transformation at scale

clang.llvm.org/extra/clang-tidy/checks/modernize-pass-by-value

Myth #36

Always pass input arguments
by const reference

There's even a clang-tidy *modernizer* check to perform this transformation at scale

clang.llvm.org/extra/clang-tidy/checks/modernize-pass-by-value

Myth #37

Make All Data Members Private ?

Myth #37

Make All Data Members Private ?

- typically seen as good practice

Myth #37

Make All Data Members Private ?

- typically seen as good practice
- enforces encapsulation: the object is in control of its internal states

Myth #37

Make All Data Members Private ?

- typically seen as good practice
- enforces encapsulation: the object is in control of its internal states
- not all types have invariants to enforce (document invariants)

Myth #37

Make All Data Members Private ?

- typically seen as good practice
- enforces encapsulation: the object is in control of its internal states
- not all types have invariants to enforce (document invariants)
- narrow/wide contracts

Myth #37

Make All Data Members Private ?

- typically seen as good practice
- enforces encapsulation: the object is in control of its internal states
- not all types have invariants to enforce (document invariants)
- narrow/wide contracts
- added complexity (YAGNI - "*You aren't gonna need it*")

Myth #37

Make All Data Members Private ?

- typically seen as good practice
- enforces encapsulation: the object is in control of its internal states
- not all types have invariants to enforce (document invariants)
- narrow/wide contracts
- added complexity (YAGNI - "*You aren't gonna need it*")
- write simpler classes

Myth #37

Make All Data Members Private ?

- typically seen as good practice
- enforces encapsulation: the object is in control of its internal states
- not all types have invariants to enforce (document invariants)
- narrow/wide contracts
- added complexity (YAGNI - "*You aren't gonna need it*")
- write simpler classes
- maybe you don't need constructors either { }

Myth #37

Make All Data Members Private ?

- typically seen as good practice
- enforces encapsulation: the object is in control of its internal states
- not all types have invariants to enforce (document invariants)
- narrow/wide contracts
- added complexity (YAGNI - "*You aren't gonna need it*")
- write simpler classes
- maybe you don't need constructors either { }
- refactoring concerns

Myth #37

Make All Data Members Private ?

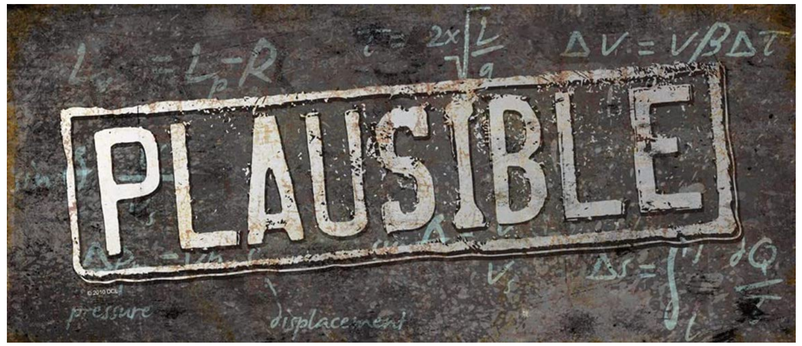
Sometimes `structs` just wanna be `structs` 😄

Myth #37


video to be published soon
ACCU 2022
ABSTRACTION PATTERNS:
MAKING CODE RELIABLY BETTER
WITHOUT DEEP UNDERSTANDING
KATE GREGORY
00:02:39 / 01:24:13
Speed
CC

Myth #37

Make All Data Members Private ?



Sometimes `structs` just wanna be `structs` 😄

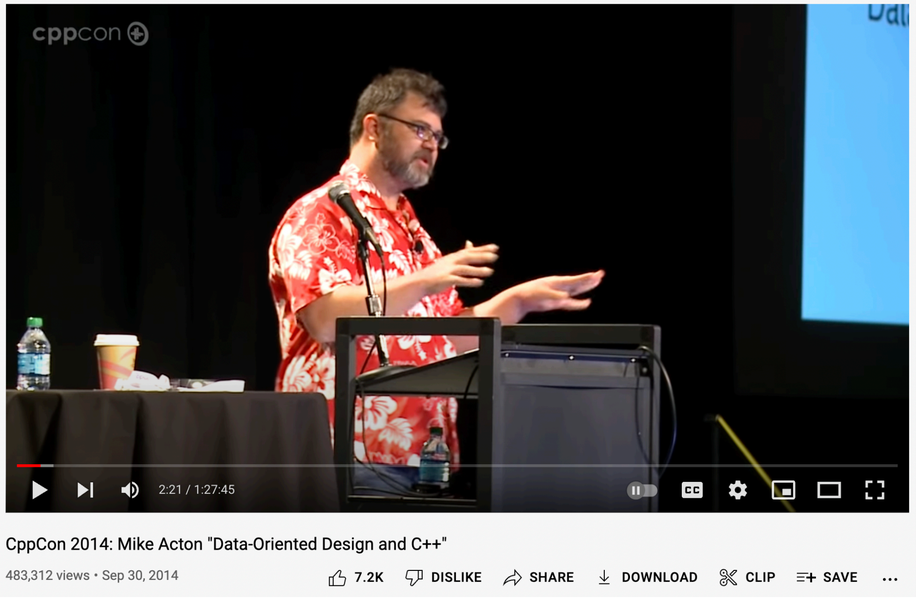
Recommended

Sometimes `structs` just wanna be `structs` 😄

While we're on the subject of `structs` ...

Recommended

Recommended



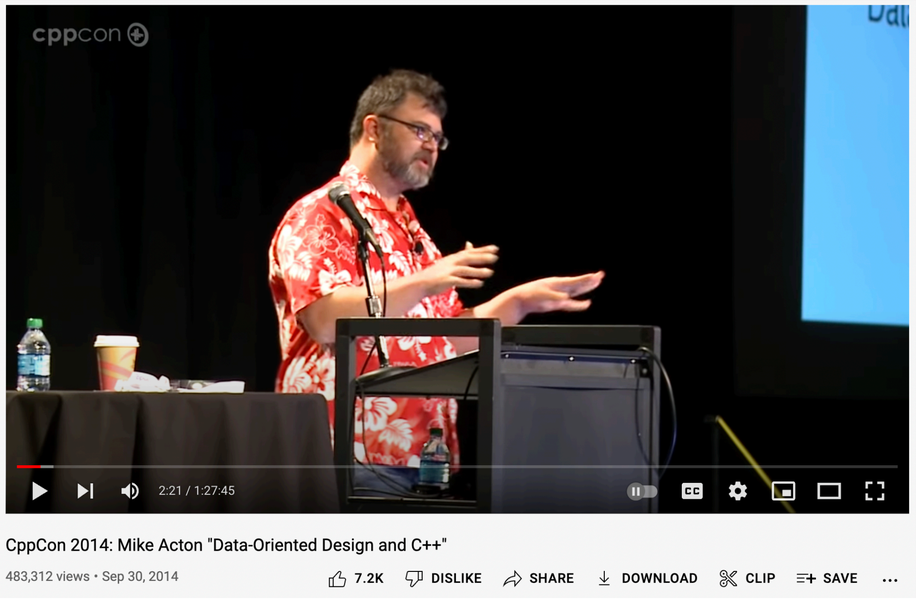
youtube.com/watch?v=rX0ItVEVjHc

Unpopular opinion (in the C++ community):

But I still think Mike Acton's keynote at CppCon 2014 is one of the best CppCon keynotes ever.

I still remember the audience gasps 😬

Recommended



youtube.com/watch?v=rX0ItVEVjHc

Unpopular opinion (in the C++ community):

But I still think Mike Acton's keynote at CppCon 2014 is one of the best CppCon keynotes ever.

I still remember the audience gasps 😬

Software = Data ➡️ xForm ➡️ Data

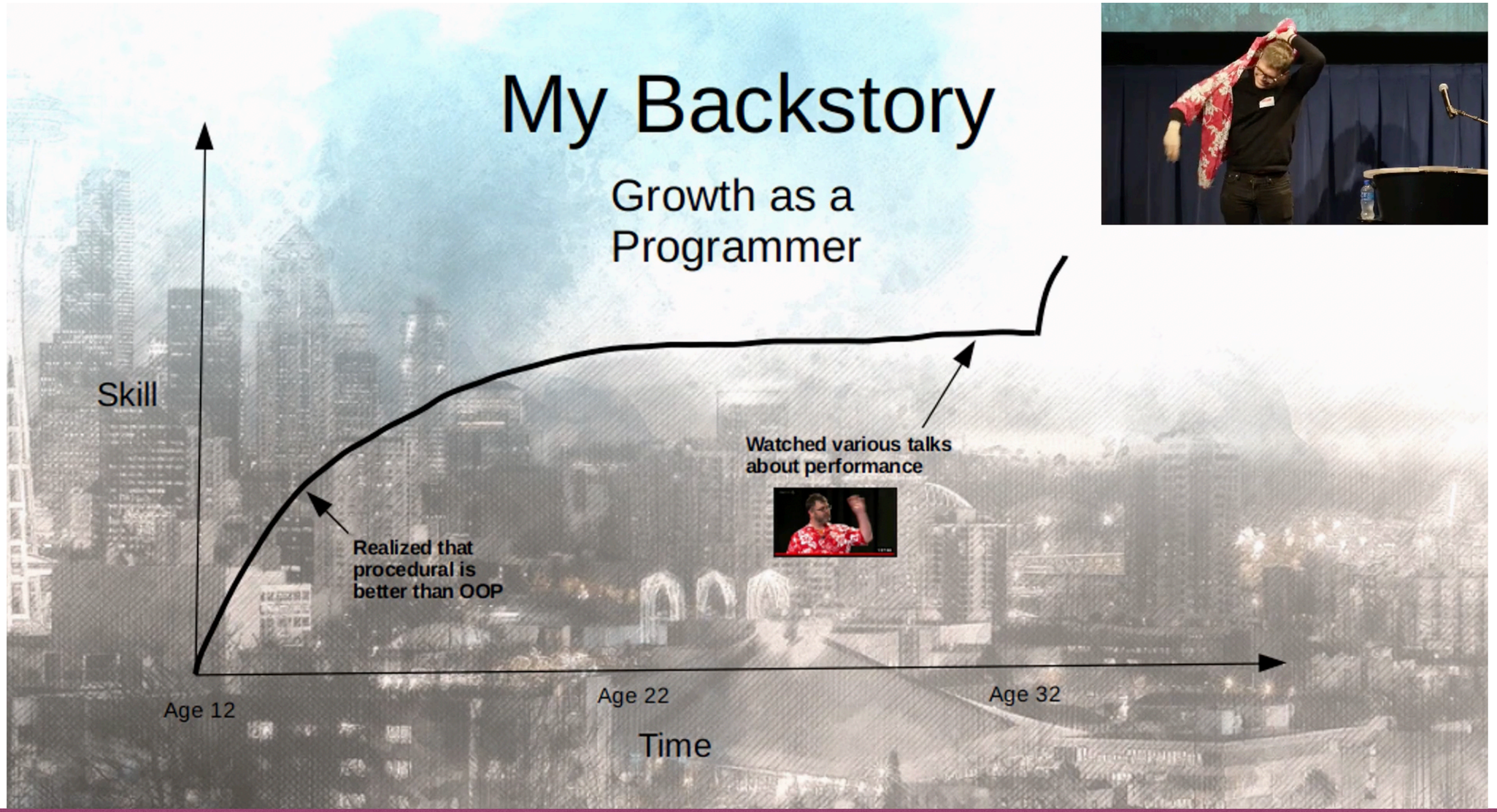
Performance = smart data layout

Memory access patterns

Recommended

A Practical Guide to Applying **Data-Oriented Design**

vimeo.com/649009599



Andrew Kelley

ZIG

Myth #40

> Iterators must go!

BoostCon 2009

"*What's up with that?*"

"*Destroy!*" 😄

I finally found a recording of this:

archive.org/details/AndreiAlexandrescuKeynoteBoostcon2009

Myth #40

> Iterators must go!

BoostCon 2009

A. Stepanov: "*Alexandrescu is on a crusade to eliminate iterators [...]*"

Programming Conversations Lecture - A9

youtube.com/playlist?list=PLHxtyCq_WDLXFAEA-lYoRNQIezL_vaSX-

Myth #40

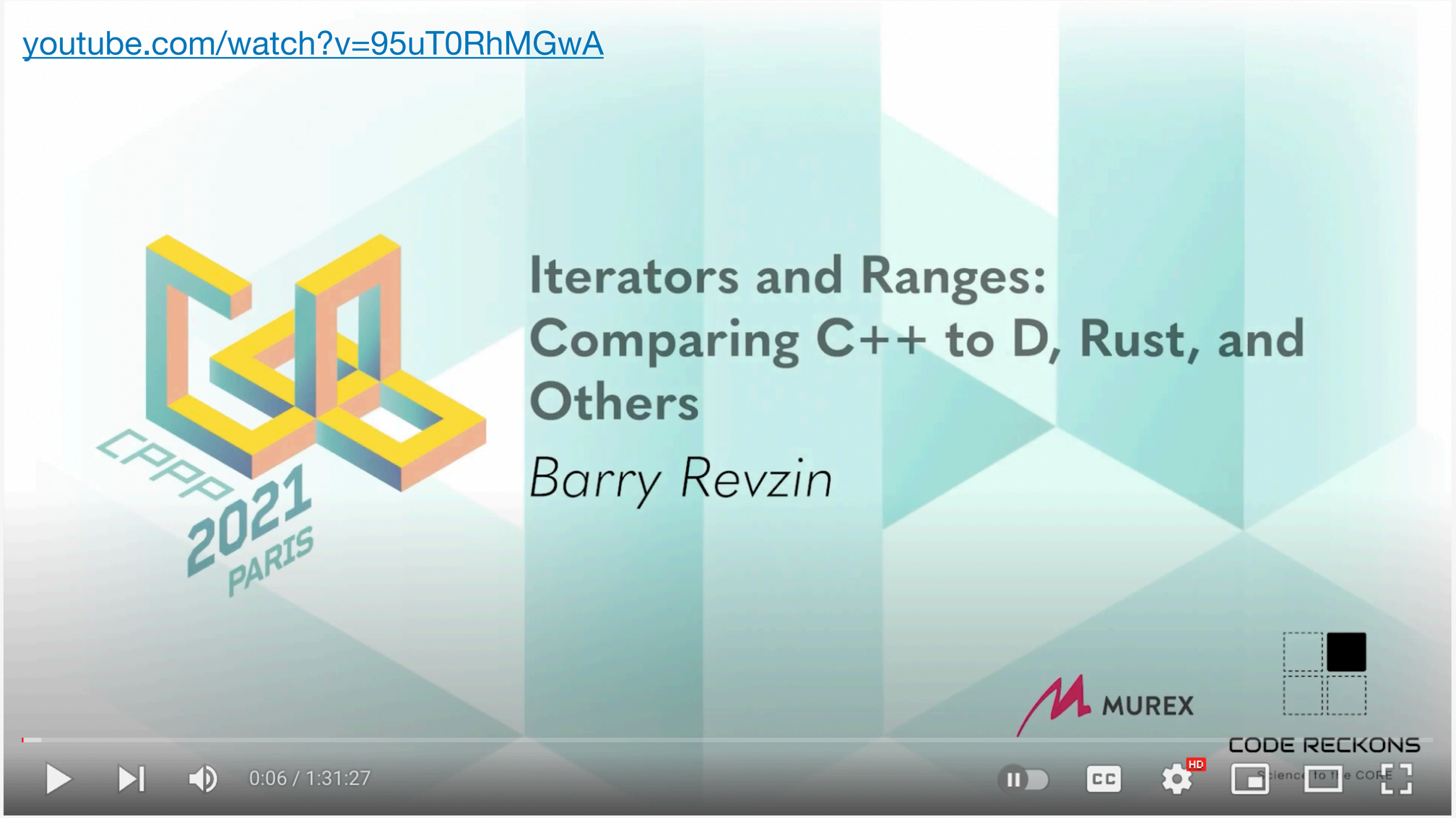
> Iterators must go!

BoostCon 2009

Someone did properly "*Destroy*", but after ~12 years...

Iterators and Ranges

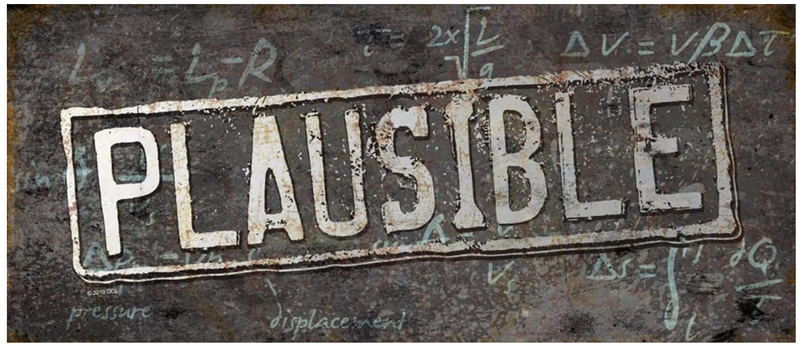
youtube.com/watch?v=95uT0RhMGwA



Keynote: Iterators and Ranges: Comparing C++ to D, Rust, and Others - Barry Revzin - CPPP 2021

Myth #40

Iterators must go!



Myth #0

New (C++) is the enemy of the old

"Before we had [feature], we were nonetheless able to program in C++"

- *Pablo Halpern*, ACCU Conf 2022 (via *Kate Gregory*)

Recommended

AURORA

Some Programming Myths Revisited - Patrice Roy - CppCon 2019 youtube.com/watch?v=KNqRjzSlUVo

Mythbusting Series

<Part 1 of N> presentation

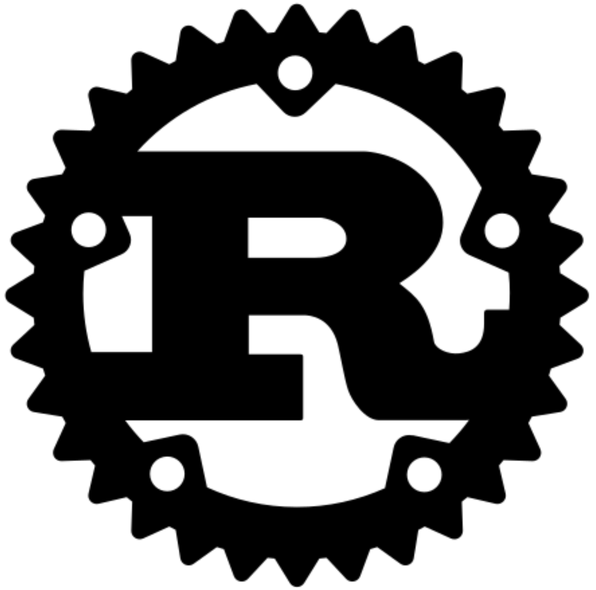
I invite other speakers to **join** this C++ Mythbusting series

Myths, in future episodes:

- strength reduction is so '90s, compilers are much smarter today
- shared_ptr is cheap (to copy)
- C++ templates produce code bloat
- C++ uniform initialization {} is to be preferred, always
- always capture [=] by value to avoid dangling
- std::string generates a lot of waste because of SSO (COW could be better)
- (N)RVO is always guaranteed
- concept checking increases build times
- function default argument values are evil
- ...

Myths, in future episodes:

- Featuring guest appearances by **Rust**
- Borrowing Trouble: the difficulties of a C++ borrow checker
- Rust and C++ interoperability
- Memory safety in Chromium
- Lifetime annotations for C++ (LLVM RFC)
- ...



ACCU 2022

@ciura_victor

Victor Ciura
Principal Engineer

CAPHYON