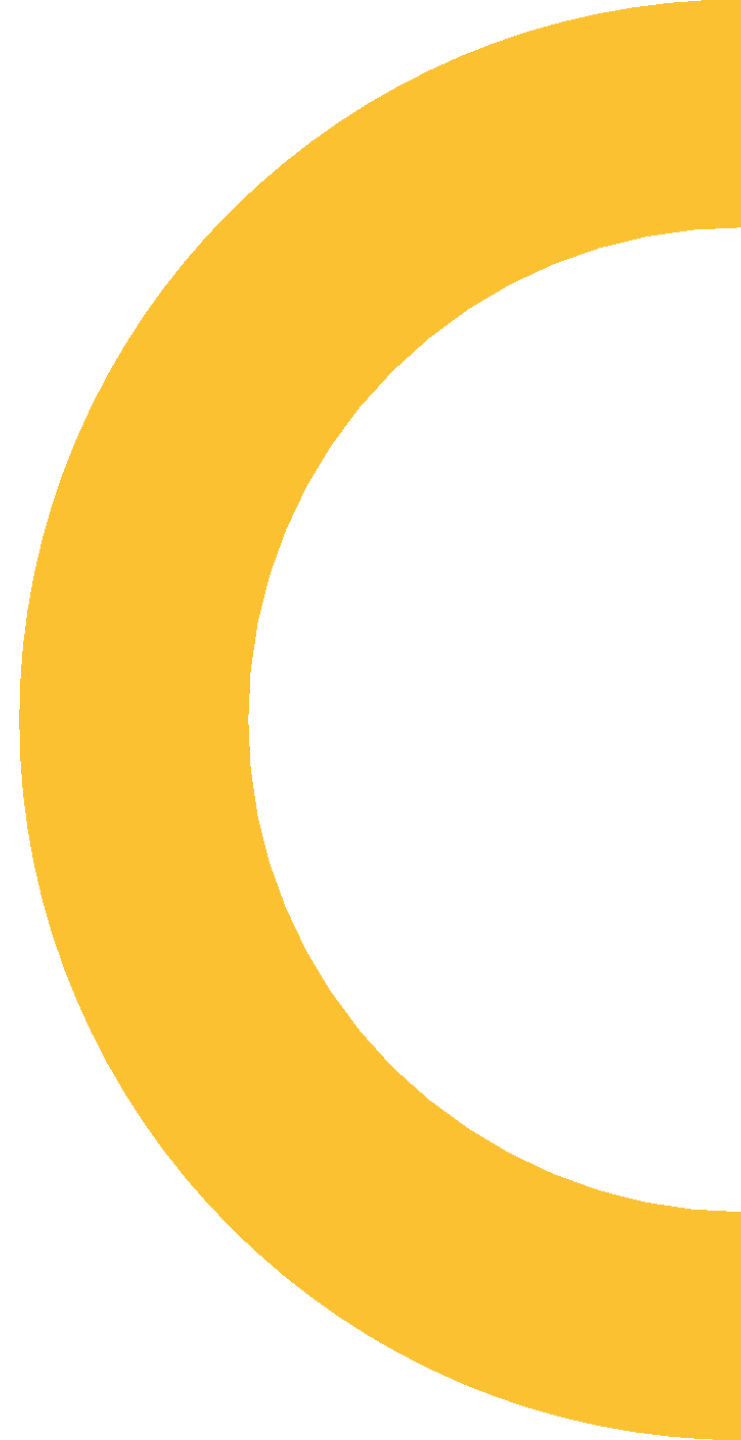




If structured binding

Zhihao Yuan

2023/11/7



Show to_chars without this proposal



```
std::array<char, 10> str;

if (auto res = std::to_chars(str.data(), str.data() + str.size(),
                             format_args...))
    std::println("{} ", std::string_view(str.data(), res.ptr));
else
    std::println("{} ", std::make_error_code(res.ec).message());
```

Show to_chars with this proposal



```
std::array<char, 10> str;

if (auto [ptr, ec] = std::to_chars(str.data(), str.data() + str.size(),
                                   format_args...))
    std::println("{} ", std::string_view(str.data(), ptr));
else
    std::println("{} ", std::make_error_code(ec).message());
```



Examples

Example 1

```
if (auto [first, last] = parse(begin(), end()); first != last)
{
    // interpret [first, last) into a value
}
```

Why don't I write
first != last &&
std::string_view(first, last)
!= KW_ANY
?

Does this mean
last != end()
?

```
struct parse_window  
{  
    char const *first, *last;  
};
```

```
parse_window parse(char const*, char const*);
```

Adding operator bool

```
struct parse_window
{
    char const *first, *last;
    explicit operator bool() const noexcept
    {
        return first != last;
    }
};

parse_window parse(char const*, char const*);
```

Example 1

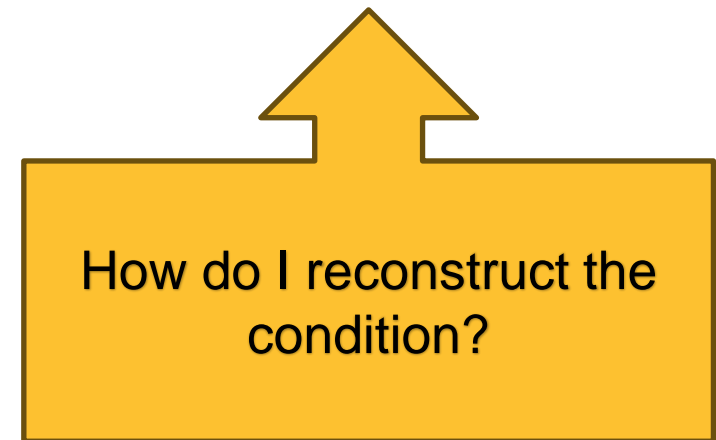
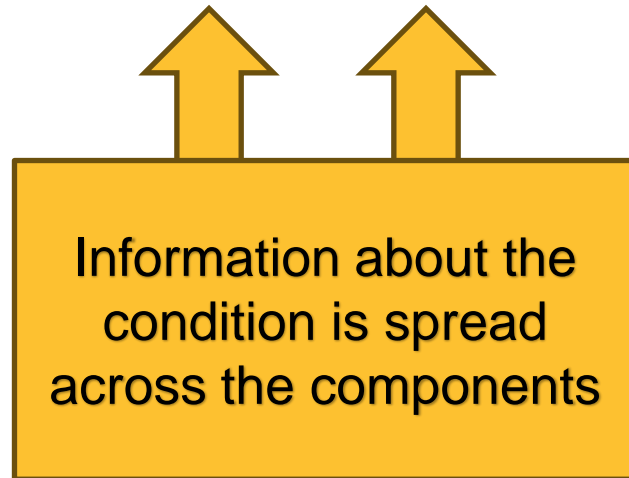
```
if (auto [first, last] = parse(begin(), end())); first != last)
{
    // interpret [first, last) into a value
}
```


Example 1 simplified with this proposal

```
if (auto [first, last] = parse(begin(), end()))  
{  
    // interpret [first, last) into a value  
}
```

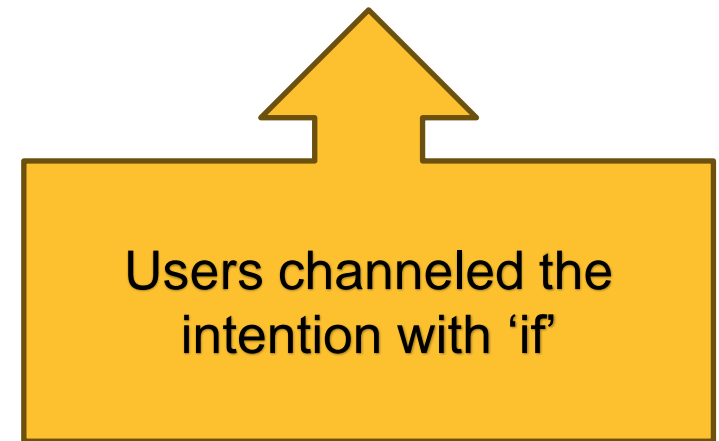
Example 1

```
if (auto [first, last] = parse(begin(), end()); first != last)  
{
```



Example 1 simplified with this proposal

```
if (auto [first, last] = parse(begin(), end()))  
{
```



Example 2...?

```
if (auto result = std::to_chars(p, last, 42))  
{  
    // okay to proceed  
}
```

Why don't you...?

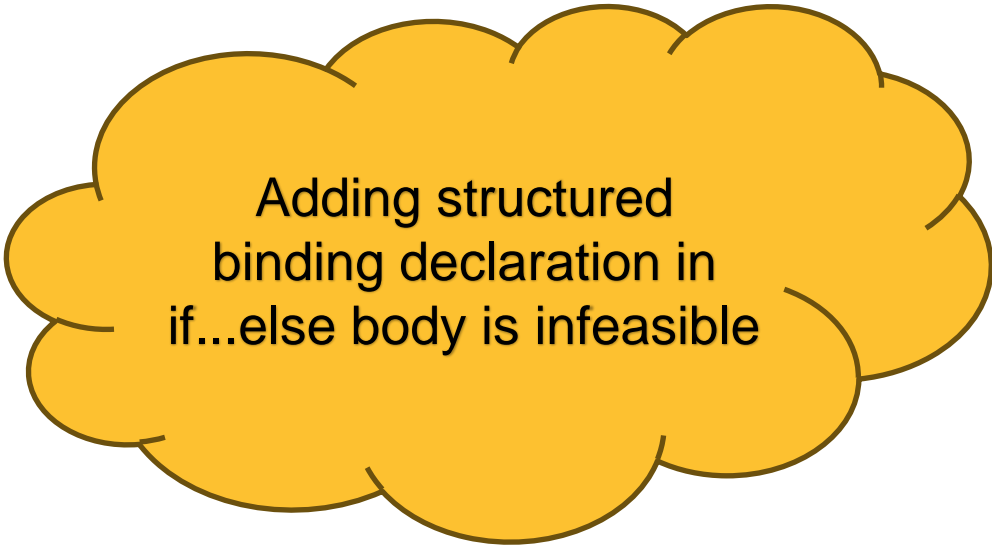
```
if (auto result = std::to_chars(p, last, 42))  
{  
    auto [ptr, ec] = result;  
    // okay to proceed  
}
```

Example 2

```
if (auto result = std::to_chars(p, last, 42))
{
    // okay to proceed
}
else
{
    // handle errors
}
```

Example 2

```
if (auto result = std::to_chars(p, last, 42))
{
    auto [ptr, _] = result;
    // okay to proceed
}
else
{
    auto [ptr, ec] = result;
    // handle errors
}
```



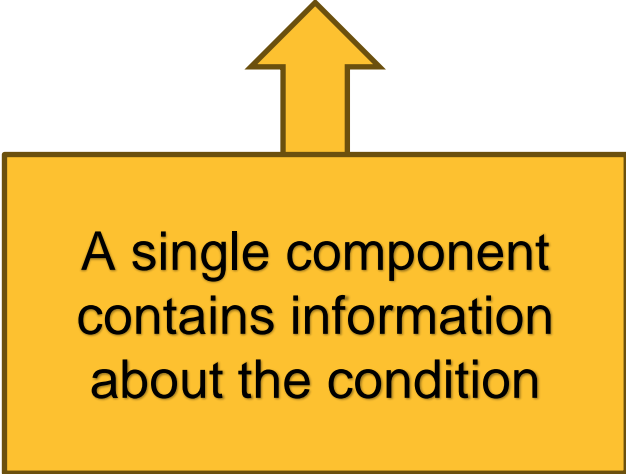
Adding structured
binding declaration in
if...else body is infeasible

Why don't you...?

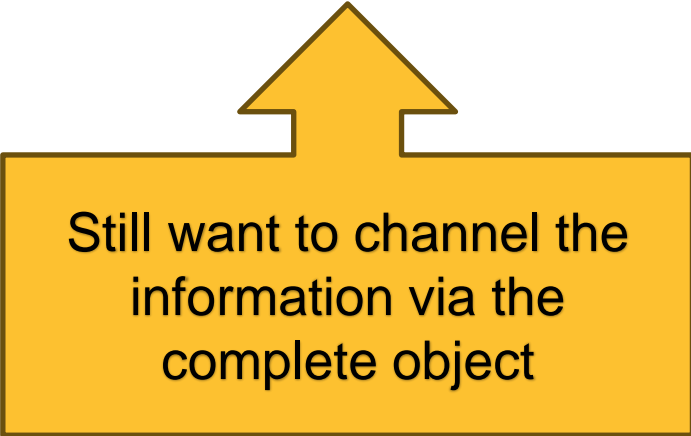
```
if (auto [ptr, ec] = std::to_chars(p, last, 42); ec != std::errc{})  
{  
    // okay to proceed  
}  
else  
{  
    // handle errors  
}
```


Why don't you...?

```
if (auto [ptr, ec] = std::to_chars(p, last, 42); ec != std::errc{})  
{
```



A single component
contains information
about the condition



Still want to channel the
information via the
complete object

Between this

```
if (auto [ptr, ec] = std::to_chars(p, last, 42); ec != std::errc{})  
{  
    // okay to proceed  
}  
else  
{  
    // handle errors  
}
```

...and this

```
if (auto result = std::to_chars(p, last, 42))
{
    auto [ptr, _] = result;
    // okay to proceed
}
else
{
    auto [ptr, ec] = result;
    // handle errors
}
```

I choose

```
if (auto [ptr, ec] = std::to_chars(p, last, 42))
{
    // okay to proceed
}
else
{
    // handle errors
}
```

Example 3

iterative solver

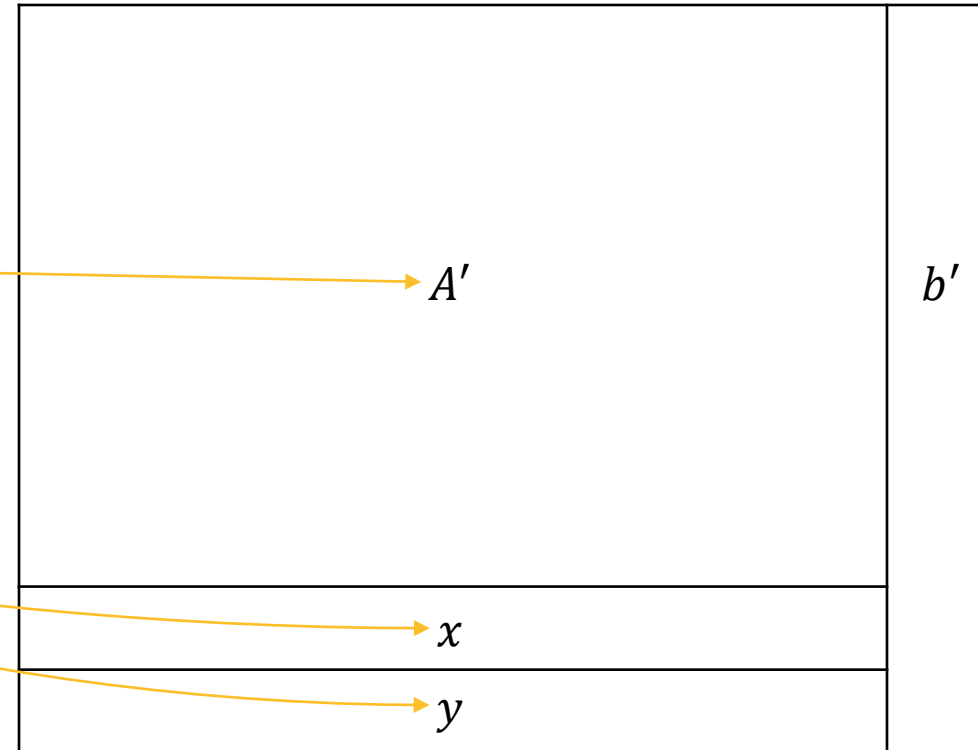
```
struct tableau
```

```
{
```

```
    matrix Ap;
```

```
    vector bp, x, y;
```

```
};
```



Example 3

iterative solver

```
struct tableau
{
    matrix Ap;
    vector bp, x, y;
};
```

```
class Solver
{
public:
    tableau solve();
    bool is_optimal(vector const&);
};
```

Example 3

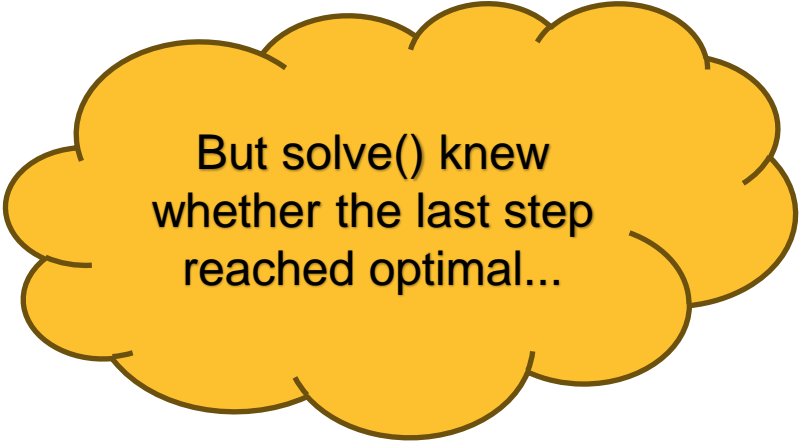
```
for (;;)
{
    // ...
    auto [Ap, bp, x, y] = solve();

    // stop iteration
}
```

Example 3

```
for (;;)
{
    // ...
    auto [Ap, bp, x, y] = solve();


    if (is_optimal(x))
        break;
}
```



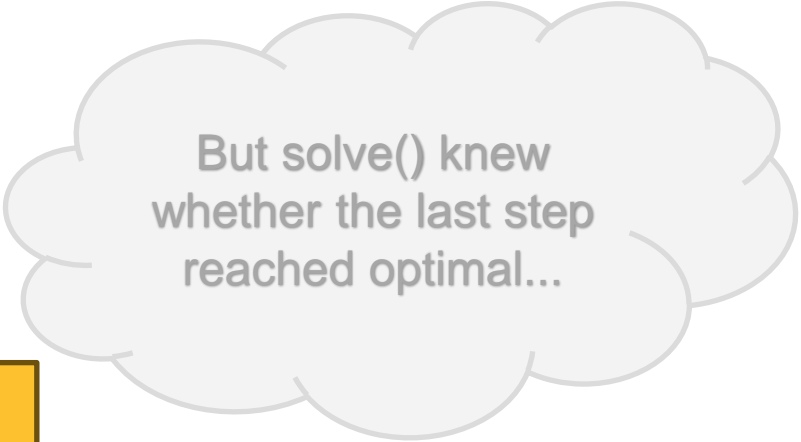
But solve() knew
whether the last step
reached optimal...

Example 3 improved with the proposal

```
for (;;)
{
    // ...
    if (auto [Ap, bp, x, y] = solve())
        break;
}
```



Saves the cost of
reconstructing the information
from the components



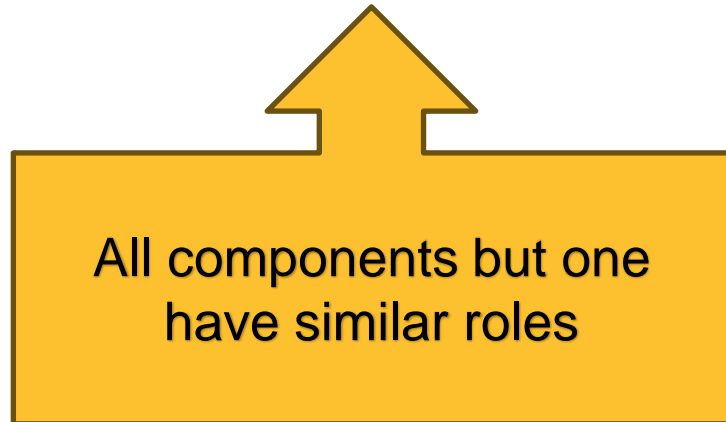
But solve() knew
whether the last step
reached optimal...

Example 4

```
if (auto [all, city, state, zip] =  
    ctref::match<"(\\w+), (\\w+) (\\d+)">(s); all)  
{  
    return location{ city, state, zip };  
}
```

Example 4

```
if (auto [all, city, state, zip] =  
    ctref::match<"(\\w+), (\\w+) (\\d+)">(s); all)  
{
```



All components but one
have similar roles

Example 4 designed differently with the proposal



```
if (auto [city, state, zip] = ctre2::match<"(\\w+), (\\w+) (\\d+)">(s))
{
    return location{ city, state, zip };
}
```

A diagram consisting of three yellow curved arrows pointing from the capture groups in the regular expression to the corresponding variables in the C++ code. The first arrow points from the first "(\\w+)" group to the "city" variable. The second arrow points from the second "(\\w+)" group to the "state" variable. The third arrow points from the "(\\d+)" group to the "zip" variable.



Technical Details

Where can I use this?

- `if constexpropt (init-statementopt condition) statement else statement`
- `switch (init-statementopt condition) statement`
- `while (condition) statement`
- `for (init-statement conditionopt ; expressionopt) statement`

Can decomposition be conditional?

```
auto consume_int() -> std::optional<int>;
```

```
if (auto [i] = consume_int()) // let e be the underlying object
{
    // i = *e
}
else
{
    // *e is not evaluated
}
```


Can decomposition be conditional?

- No

To destructure	Abusing condition
<code>optional<T></code>	<code>[x]</code>


Can decomposition be conditional?

- No

To destructure	Abusing condition
<code>optional<T></code>	<code>[x]</code>
<code>optional<tuple<T, U>></code>	<code>[x, y]</code> 

Can decomposition be conditional?

- No

To destructure	Abusing condition	Pattern matching
<code>optional<T></code>	<code>[x]</code>	<code>let ?x</code>
<code>optional<tuple<T, U>></code>	<code>[x, y]</code> 	<code>let ?[x, y]</code>

Pattern matching's solution

```
if (consume_int() match let ?i)
{
    // use(i)
}
else
{
    // has no value
}
```

When does the Boolean test happen?

- After initializing the bindings
- So that

```
if (auto [a, b, c] = fn())  
{  
    statements;  
}
```

is equivalent to

```
if (auto [a, b, c] = fn(); underlying-object)  
{  
    statements;  
}
```

Can I decompose an array in a condition?



- No.



Implementation Experience

Implemented in Clang



```
A ▾ 🗄️ + ▾ 📄 🔍 🗑️ C++ ▾
9
10 struct format_status
11 {
12     format_errc ec;
13     char *bp;
14
15     explicit operator bool() const noexcept
16     {
17         return ec == format_errc::no_error;
18     }
19 };
20
21 format_status
22
23 int main()
24 {
25     if (auto [ok, ptr] = readint())
26     {
27         printf("stopped at %p\n", ptr);
28     }
29 }
```

warning: ISO C++17 does not permit structured binding declaration in a condition [-Wbinding-in-condition] x86-64 clang (trunk) #1
[View Problem \(Alt+F8\)](#) No quick fixes available

```
x86-64 clang (trunk) ▾ 🗄️ + ▾ 🔍 🗑️ -Wall -std=c++17
A ▾ ⚙️ ▾ 🗄️ 🔍 + ▾ 🗑️
1 main: # @main
2     push rax
3     call readint()@PLT
4     test eax, eax
5     jne .LBB0_2
6     lea rdi, [rip + .L.str]
7     mov rsi, rdx
8     xor eax, eax
9     call printf@PLT
10 .LBB0_2:
11     xor eax, eax
12     pop rcx
13     ret
14 .L.str:
15     .asciz "stopped at %p\n"
```



Thank you