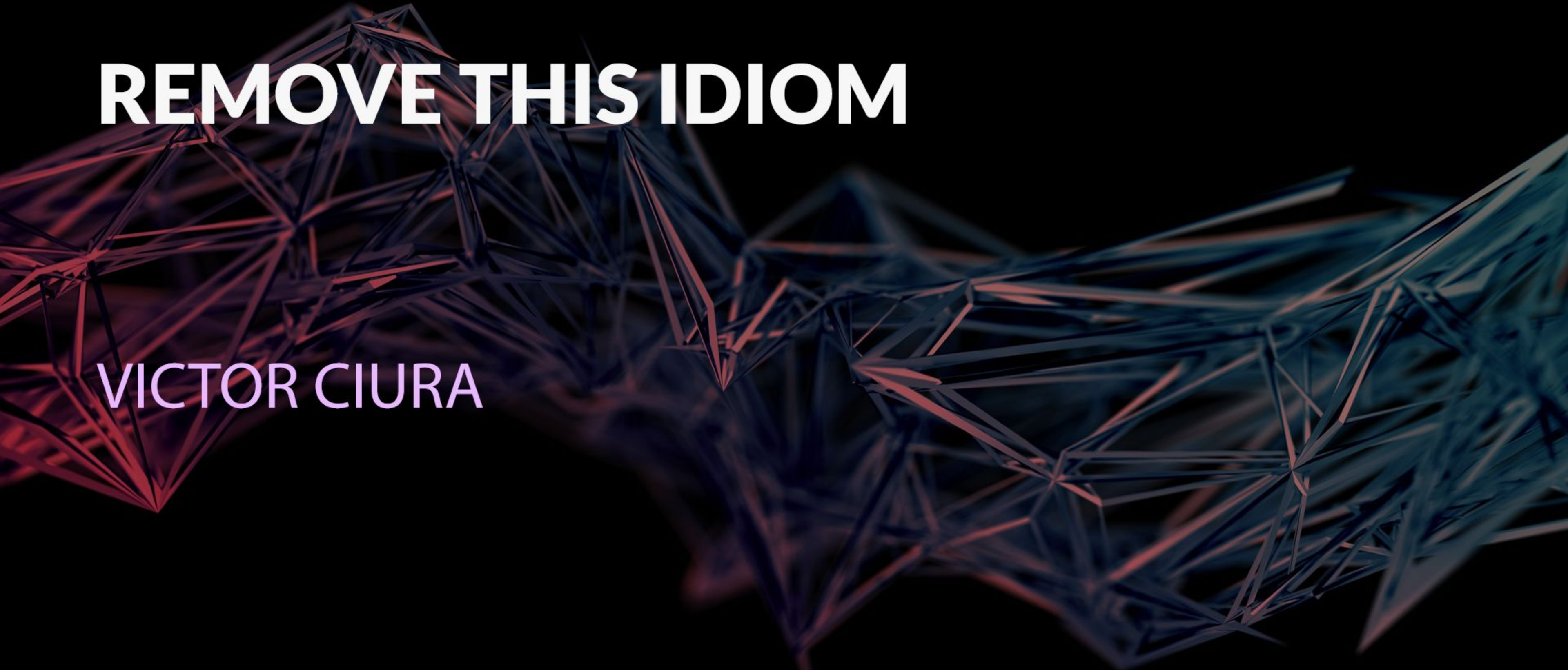
accu 2022

REMOVE THIS IDIOM

VICTOR CIURA



Abstract

Most seasoned C++ developers are familiar with the erase-remove(_if) idiom when using STL. The std::remove(_if) algorithm is a showcase of the beauty of STL generic design and the power of algorithmic composition through iterators.

Yet this seemingly trivial task of removing elements from a collection (based on a predicate) is an epitome of C++ footguns. And no modern compiler or static analysis can warn you about it. We are on our own. Ask me how I know…

Fear not, C++20 has a better alternative for us and it’s 100% safe and much leaner to use.

Remove This Idiom

accu 2022

@ciura_victor

Victor Ciura
Principal Engineer

CAPHYON

Advanced Installer

Clang Power Tools

@ciura_victor

`[[nodiscard]]`❤️

`[[nodiscard]]`❤️

MSVC https://github.com/microsoft/STL/issues/206 + marked over 3,000 functions

```
warning C4834:
discarding return value of function with 'nodiscard' attribute
```

Compiler toolset upgrades ⚙️
can be a hassle and a blessing

`[[nodiscard]]`❤️

MSVC https://github.com/microsoft/STL/issues/206

```
warning C4834:
discarding return value of function with 'nodiscard' attribute
```

Clang libc++ 🤐

GCC libstdc++ 🤐

https://godbolt.org/z/odcrGf4rn

A simple task...

Remove elements matching a predicate.

Given:

```
std::vector<int> v = { 1, 2, 3, 4, 5, 6, 7 };
```

How do we remove all **even** numbers ?

Write a quick `for()` loop ? ❌ (10x engineer)

Use `<algorithm>` ? ✅ (galaxy brain)

Remove elements matching a predicate.

https://en.cppreference.com/w/cpp/algorithm/remove

```
template< class ForwardIt, class UnaryPredicate >
ForwardIt std::remove_if(ForwardIt first, ForwardIt last, UnaryPredicate p);
```

🔥

```
warning C4834:
discarding return value of function with 'nodiscard' attribute
```

Remove elements matching a predicate.

https://en.cppreference.com/w/cpp/algorithm/remove

```
template< class ForwardIt, class UnaryPredicate >
ForwardIt std::remove_if(ForwardIt first, ForwardIt last, UnaryPredicate p);
```

🔥 Where are your unit tests*, buddy ?

And why were they passing ?

* subject for another presentation :)

Remove elements matching a predicate.

https://en.cppreference.com/w/cpp/container/vector/erase

```
iterator vector::erase(const_iterator pos);
iterator vector::erase(const_iterator first, const_iterator last);
```

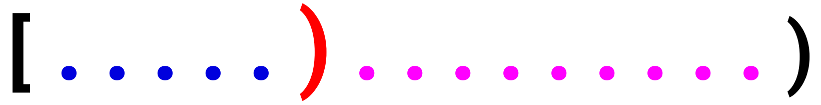
The forgotten friend 😱

Remove elements matching a predicate.

https://en.cppreference.com/w/cpp/algorithm/remove

```
template< class ForwardIt, class UnaryPredicate >
ForwardIt std::remove_if(ForwardIt first, ForwardIt last, UnaryPredicate p);
```

- moves elements around, based on the given predicate
- returns past-the-end iterator for the new range of values:



- does not change the **size** of the container!!!

Remove IF Algorithm

```
std::vector<int> v = { 1, 2, 3, 4, 5, 6, 7 };

auto pos = std::remove_if(v.begin(), v.end(),
                                [] (int i) { return (i & 1) == 0; });
```

Q:
How do you think this works ?

A:

“`remove_if()` moves all the elements you want to remove to the **end** of the vector,

then the `erase()` gets rid of them.”

```
v = { 1, 3, 5, 7, 2, 4, 6 }
```

WRONG !

Remove IF Algorithm

```
std::vector<int> v = { 1, 2, 3, 4, 5, 6, 7 };
```

```
auto pos = std::remove_if(v.begin(), v.end(),
                          [] (int i) { return (i & 1) == 0; });
```

This **isn’t** what `std::remove_if()` does !!!

If it did that – which is ***more work*** than it needs – it would in fact be `std::partition()`

What `std::remove_if()` does is move the elements that **won’t** be removed **to the beginning**.

The algorithm cares only about the elements we want to **keep**.

Remove IF Algorithm

```
std::vector<int> v = { 1, 2, 3, 4, 5, 6, 7 };
```

```
auto pos = std::remove_if(v.begin(), v.end(),
                           [] (int i) { return (i & 1) == 0; });
```

What about the elements at the **end** of the vector ?

GARBAGE !

They get *overwritten* in the process of `std::remove()` algorithm.

```
v = { 1, 3, 5, 7, 5, 6, 7 }
                 ⇧
```

`where the iterator **returned** by remove_if() points`

Erase-Remove Idiom

```
std::vector<int> v = { 1, 2, 3, 4, 5, 6, 7 };
```

```
v.erase( std::remove_if(v.begin(), v.end(),
                               [] (int i) { return (i & 1) == 0; }),
         v.end() );
```

Erase the (garbage) elements at the **end** of the vector ?

Before `erase()` is called: v = { 1, 3, 5, 7, 5, 6, 7 }

where the iterator **returned** by remove_if() points

After `erase()` is called: v = { 1, 3, 5, 7 }

Erase-Remove Idiom

```
std::vector<int> v = { 1, 2, 3, 4, 5, 6, 7 };

v.erase( std::remove_if(v.begin(), v.end(),
                        [] (int i) { return (i & 1) == 0; }),
         v.end() );
```

```
iterator vector::erase(const_iterator first, const_iterator last);
```

the iterator **returned** by remove_if()

```
iterator vector::erase(const_iterator pos);
```

Erase-Remove Idiom

```
std::vector<int> v = { 1, 2, 3, 4, 5, 6, 7 };

v.erase( std::remove_if(v.begin(), v.end(),
                           [] (int i) { return (i & 1) == 0; }),
         v.end() );
```

A very forgettable `end()` that will ***silently*** select the wrong `erase()` overload:

```
iterator vector::erase(const_iterator pos);
```

This will erase just a single element from the vector - **Oops!** not what we intended :(

Erase-Remove Idiom

```
std::vector<int> v = { 1, 2, 3, 4, 5, 6, 7 };

v.erase( std::remove_if(v.begin(), v.end(),
                        [] (int i) { return (i & 1) == 0; }) );
```

A very forgettable `end()` that will ***silently*** select the wrong `erase()` overload:

```
iterator vector::erase(const_iterator pos);
```

This will erase just a single element from the vector - **Oops!** not what we intended :(

Erase-Remove-End Idiom

```
std::vector<int> v = { 1, 2, 3, 4, 5, 6, 7 };

v.erase( std::remove_if(v.begin(), v.end(),
                        [] (int i) { return (i & 1) == 0; }),
         v.end() );
```

That’s a mouthful… and we’re not very good with idioms/acronyms in C++

E.R.E.I. 😄

Erase IF

C++20

All-in-one C++20 solution:

https://en.cppreference.com/w/cpp/container/vector/erase2

https://en.cppreference.com/w/cpp/container/list/erase2

```
template< class T, class Alloc, class Pred >
constexpr typename std::vector<T,Alloc>::size_type
    erase_if(std::vector<T,Alloc> & c, Pred pred);
```

```
template< class T, class Alloc, class Pred >
typename std::list<T,Alloc>::size_type
    erase_if(std::list<T,Alloc> & c, Pred pred);
```

Erase IF

C++20

All-in-one `C++20` solution:

https://en.cppreference.com/w/cpp/container/vector/erase2

https://en.cppreference.com/w/cpp/container/list/erase2

```
std::vector<int> v = { 1, 2, 3, 4, 5, 6, 7 };

std::erase_if(v, [] (int i) { return (i & 1) == 0; });
```

That's it 🎉

The equivalent of doing the Erase-Remove Idiom, but shorter & safer

Let’s remove this Erase-Remove Idiom, for good

```
v.end()
```

Remove This Idiom

accu 2022

@ciura_victor

Victor Ciura
Principal Engineer

CAPHYON