C++20 Ranges

David Sullins

St. Louis C++ meetup

June 12, 2019

Range libraries

- Boost.Range
- range-v3
  - Eric Niebler’s library, the most featureful
  - Doesn’t require C++20 concepts
  - https://github.com/ericniebler/range-v3
- cmcstl2
  - Implementation of P0896R4 “The One Ranges Proposal”
  - Heavily tied to C++20 concepts
  - https://github.com/CaseyCarter/cmcstl2

Range libraries

- Boost.Range
- range-v3
  - Eric Niebler’s library, the most featureful
  - Doesn’t require C++20 concepts
  - https://github.com/ericniebler/range-v3
- cmcstl2
  - Implementation of P0896R4 “The One Ranges Proposal”
  - Heavily tied to C++20 concepts
  - https://github.com/CaseyCarter/cmcstl2

Compiler Explorer

https://godbolt.org/

Examples in this presentation work with these compiler explorer settings:

- Compiler: x86-64 gcc 9.1
- Options: -std=c++2a -I/opt/compiler-explorer/libs/cmcstl2/include -fconcepts

May also work as far back as gcc 7.1. Won’t work with clang.

What’s a range?

A range is just a begin/end pair

- Anything with begin() and end() is a range
- Could be member functions or free functions in the same namespace
- begin() returns an iterator
- end() returns a sentinel

What’s a range?

A range is just a begin/end pair

- Anything with begin() and end() is a range
- Could be member functions or free functions in the same namespace
- begin() returns an iterator
- end() returns a sentinel
- They don’t have to return the same types!

What’s a range?

A range is just a begin/end pair

- Anything with begin() and end() is a range
- Could be member functions or free functions in the same namespace
- begin() returns an iterator
- end() returns a sentinel
- They don’t have to return the same types!

;

What’s a range?

A range is just a begin/end pair

C++20 ranges are **so much more** than just a begin/end pair

What's a range?

A range is just a begin/end pair

C++20 ranges are **so much more** than just a begin/end pair

Most standard library algorithms have versions that accept ranges instead of begin/end pairs in C++20

```
std::ranges::sort(v);      vs      std::sort(v.begin(), v.end());
```

What's a range?

A range is just a begin/end pair

C++20 ranges are **so much more** than just a begin/end pair

Most standard library algorithms have versions that accept ranges instead of begin/end pairs in C++20

```
std::ranges::sort(v);      vs      std::sort(v.begin(), v.end());
```

And there's more: they improved the entire set of STL algorithms

Compiler explorer

- Range for https://godbolt.org/z/QTgjtX
- Range begin https://godbolt.org/z/feXDqv
- Range foreach https://godbolt.org/z/_--SPj
- Range projections https://godbolt.org/z/LV_K2k
- Range generators https://godbolt.org/z/7v7NSv
- Range adaptors https://godbolt.org/z/jNiiPo
- Range comprehensions https://godbolt.org/z/94NzTL

Views vs Containers

- View conceptually is a range of references
- Container owns its elements, view does not
- Container deep copies its elements, view does not: O(n) vs O(1)
- But, we can still do many of the same operations on a view as a container
- Operations on containers are eagerly evaluated
- Operations on views are often lazily evaluated

Recap

- A range is a begin/end pair
- C++20 ranges library improves much of the standard library
- Views are lightweight ranges that support lazy evaluation and some functional programming paradigms
- Just replace std:: with std::ranges:: for better versions of most STL algorithms

Where to learn about ranges

These are all good references that I used for this talk

- https://en.cppreference.com/w/cpp/ranges
  - Documentation incomplete
- https://wg21.link/p0896r4
  - Changes to the C++ working draft for ranges
- https://github.com/CaseyCarter/cmcstl2
  - I found searching through the code to be very helpful
- Eric Niebler's writings and talks
  - http://ericniebler.com/2018/12/05/standard-ranges/

Email: david.sullins@gmail.com

Mastodon: @davidsullins@mastodon.gamedev.place

Bonus: monads in C++

Stolen from Bartosz Milewski's blog

https://bartoszmilewski.com/2014/10/17/c-ranges-are-pure-monadic-goodness/

Functor

In math and in Haskell “functor” means something different than how we use it in C++

- Generic template that allows “lifting” of functions
- Generic template: can be instantiated for any type
  - Range is a generic template, can have a range of anything including other ranges
- Lifting: apply function T -> U to a range of T, get a range of U
  - view::transform for example

view::iota(1) | view::transform(times3) | view::take(10)

Pointed Functor

Pointed functor is a functor that lets you lift individual values

- view::single

Applicative functor

Applicative functor is a pointed functor that lets you lift multi-argument functions

- view::transform lifts single-argument functions
- How do we lift multi-argument functions?

todo

Monad

Monad is an applicative functor an additional ability: a way of flattening a doubly encapsulated object

- That for_each function in the range comprehension example is called a “monadic bind”
- C++20 ranges are monads

Monad

Monad is an applicative functor an additional ability: a way of flattening a doubly encapsulated object

- That for_each function in the range comprehension example is called a “monadic bind”
- C++20 ranges are monads

Don’t worry, I still don’t know what a monad is either