Go, Go Power Ranges!

szczecin::cpp #1

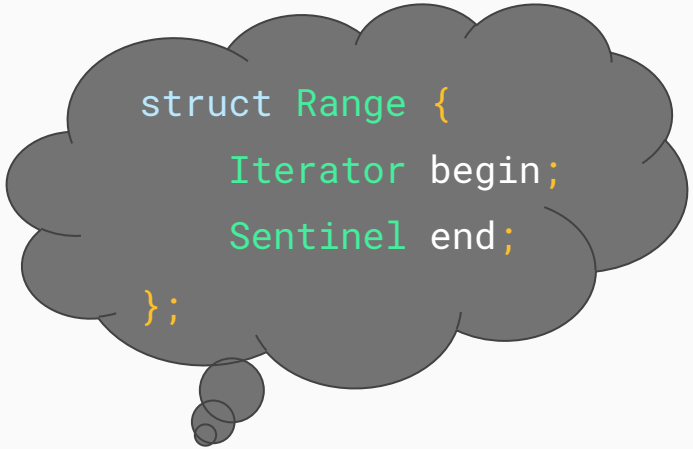
2019-03-14, Michał Kucharski, Lead Software Engineer at GlobalLogic

The question

Which programming concept
can make your software more powerful?

Ranges - introduction

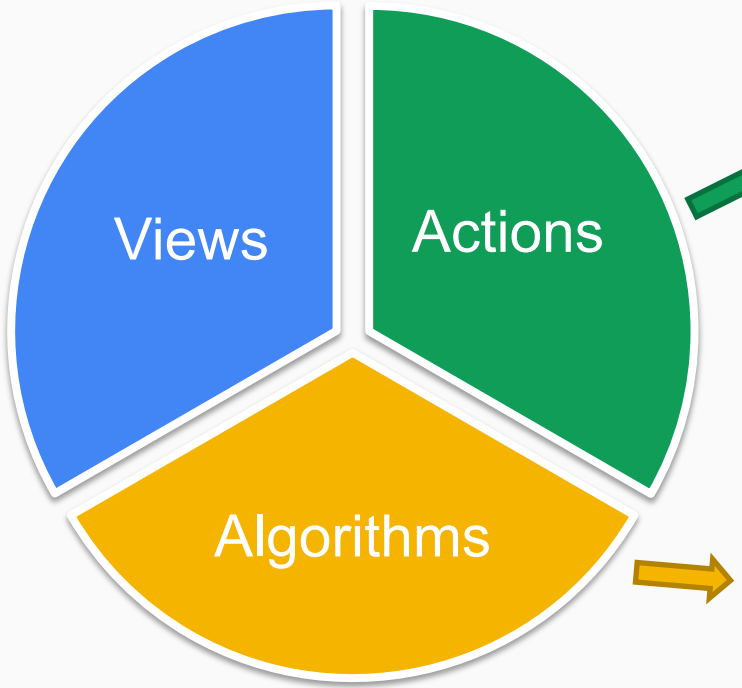
- A ***range*** can be thought of as a *pair of iterators*.
- Keeping two iterators into a single object simplifies a lot of things.
- In C++20 `std::ranges` will extend standard library with algorithms and facilities to work with range objects.



https://ericniebler.github.io/range-v3/

What's inside the ranges library?

Lazy,
non-owning,
non-mutating,
composable
range views



Eager, mutating,
composable
algorithms

Iterator- and range-based algorithms, like the standard algorithms (with cool stuff like projections)

History

- Range v1 (2004)
  - In Boost; Simple overloads for containers, e.g. `boost::sort(x);`
- Range v2 (2010)
  - In Boost; filter/transform adaptors, syntactic sugar - pipe operator |
- Range v3 (2013)
  - Update for C++11 (rvalue aware; views/actions/algorithms)
- Standardization (2018)
  - Ranges merged into C++20 (`std::ranges`)

Quiz time (1)

https://menti.com

Show me the code

```
// Remove duplicates and remove the numbers that are even

// BEFORE:
std::vector<int> vec = read_integers();

std::sort(vec.begin(), vec.end());
vec.erase(std::unique(vec.begin(), vec.end()), vec.end());
vec.erase(std::remove_if(vec.begin(), vec.end(), [](auto i){ return i % 2 == 0; }), vec.end());
```

```
//AFTER:
std::vector<int> vec = read_integers()
                     | std::ranges::action::sort
                     | std::ranges::action::unique
                     | std::ranges::action::remove_if([](auto i){ return i % 2 == 0; });
```

Show me more code

```
// Print first 3 integers smaller than let's say 42

// BEFORE:
std::vector<int> vec {/*...*/};

int printed_count = 0;
for (auto i : vec) {
    if (i < 42) {
        std::cout << i << " ";
        printed_count++;
        if (printed_count == 3) {
            break;
        }
    }
}
```

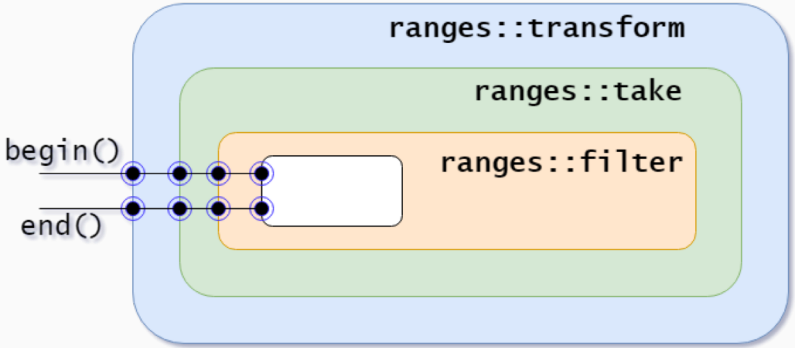
```
// AFTER:
std::vector<int> vec {/*...*/};

for (auto i : vec
    | std::ranges::view::filter([](auto e){ return e < 42; })
    | std::ranges::view::take(3))
{
    std::cout << i << " ";
}
```

- More readability
- Less repetition
- Express intent
- Composability

Benefits

- Convenience
- Composability
  - Single object can be easily wrapped into other objects
  - Passing data through a series of combinators
  - Allow lazy evaluation (on-demand)
  - Easier to reason about



Source: https://www.reddit.com/r/gifs/comments/7ns9h0/opening_russian_nesting_dolls/

More examples

```
// Generate an infinite list of integers starting at 1,
// square them, take the first 10, and sum them

const int sum = ranges::accumulate(
    ranges::view::iota(1)                                  // infinite range: 1, 2, 3, 4, 5...
    | ranges::view::transform([](int i){ return i * i; })  // square them
    | ranges::view::take(10)                               // take the first 10
, 0);                                                      // accumulate starting with 0
```

Even more examples

```
// Find a person named "Michał" in a vector of struct instances

struct Person {
    std::string name;
};
std::vector<Person> people {/*...*/};



if (const auto it = ranges::find(people, "Michał", &Person::name);
    it != ranges::end(people))
{
    std::cout << it->name << " ";
}
```

A **projection**:

- Is a unary transformation function
- Gets applied to each element before the algorithm operates on the element

Deep dive – let’s write our own view

- `ranges::view::unique` removes only the consecutive repetitions
- Let’s create a `simple_unique_view` that doesn’t care
- Choose the base class from `view_facade<>` and `view_adaptor<>`
- Implement only the necessary subset of functions – in our case: `begin()`, `next()`, `read()`
- How to check for the uniqueness...

Deep dive – let's write our own view

```
template <class Rng>                                                                    (1)
class simple_unique_view : public ranges::view_adaptor<simple_unique_view<Rng>, Rng>    (2)
{
   friend ranges::range_access;                                                         (3)

   class adaptor : public ranges::adaptor_base                                          (4)
   {
   public:
       adaptor() = default;

       static ranges::iterator_t<Rng> begin(simple_unique_view& rng) { /* ... */ }
       void next(ranges::iterator_t<Rng>& it) { /* ... */ }
       auto read(ranges::iterator_t<Rng> it) const { /* ... */ }                        (5)
   };

   adaptor begin_adaptor() { /* ... */ }                                                (6)

 public:
   simple_unique_view() = default;                                                      (7)
   simple_unique_view(Rng&& rng)
       : simple_unique_view::view_adaptor{std::forward<Rng>(rng)}
   {
   }
};
```

To sum it up...

- All would be perfect if not for:
  - Compile times (e.g. just including the header is 5 seconds more waiting)
  - Compile errors
    - Even though there is **Concepts** layer

- Range-v3
  - https://github.com/ericniebler/range-v3
  - https://godbolt.org/
  - `#include <range/v3/all.hpp>`
- Boost Range 2.0
  - http://www.boost.org/doc/libs/1_64_0/libs/range
  - `#include <boost/range/adaptors.hpp>`

Quiz time (2)

https://menti.com

Pssst… You can win a book...

That’s it, thanks!