A view of infinite <ranges> in C++20's kitchen

René Richter

2019-03-14

C++ usergroup DRESDEN

Ranges

[ancientpoint.com, ∼1920]

[ajmadison.com]

(In)finite ranges in various programming languages

Die ganzen Zahlen hat der liebe Gott gemacht,
alles andere ist Menschenwerk.
— Leopold Kronecker (1823–1891)

(In)finite ranges in various programming languages

Die ganzen Zahlen hat der liebe Gott gemacht,
alles andere ist Menschenwerk.
— Leopold Kronecker (1823–1891)

```
[0..N] -- Haskell
[0..]  -- infinite list
```

(In)finite ranges in various programming languages

Die ganzen Zahlen hat der liebe Gott gemacht,
alles andere ist Menschenwerk.
— Leopold Kronecker (1823–1891)

```
[0..N] -- Haskell
[0..]  -- infinite list
```

```
for i in range(N): print(i)      # Python
def iota(n):                     # a generator, not a range
    while True:
        yield n
        n += 1
for i in iota(0): print(i)
```

(In)finite ranges in various programming languages

Die ganzen Zahlen hat der liebe Gott gemacht,
alles andere ist Menschenwerk.
— Leopold Kronecker (1823–1891)

```
[0..N] -- Haskell
[0..]  -- infinite list
```

```
for i in range(N): print(i)      # Python
def iota(n):                     # a generator, not a range
    while True:
        yield n
        n += 1
for i in iota(0): print(i)
```

PRIMES IN APL: $(\sim N \in N \circ . \times N)/ \leftarrow 1 \downarrow \iota N$

(In)finite ranges in various programming languages

Die ganzen Zahlen hat der liebe Gott gemacht,
alles andere ist Menschenwerk.
— Leopold Kronecker (1823–1891)

```
[0..N] -- Haskell
[0..]  -- infinite list
```

```
for i in range(N): print(i)      # Python
def iota(n):                     # a generator, not a range
    while True:
        yield n
        n += 1
for i in iota(0): print(i)
```

PRIMES IN APL: $(\sim N \in N \circ . \times N) / \leftarrow 1 \downarrow \iota N$

```
auto   finite = std::ranges::view::iota(0, N);  // C++20
auto infinite = std::ranges::view::iota(0);
```

A motivating example

```
take 5 [ x*x | x <- [0..], x `mod` 2 == 1 ]
```

A motivating example

```
take 5 [ x*x | x <- [0..], x `mod` 2 == 1 ]
```

output (https://repl.it/languages/haskell):

```
[1,9,25,49,81]
```

A motivating example

```
take 5 [ x*x | x <- [0..], x `mod` 2 == 1 ]
```

output (https://repl.it/languages/haskell):

```
[1,9,25,49,81]
```

in C++?

- hand-written loop
- ...

```cpp
// C++ std containers and algorithms:
auto odd    = [](int n) { return n%2; };
auto square = [](int n) { return n*n; };
std::vector v = { 0,1,2,3,4,5,6,7,8,9 };

std::vector<int> temp;
std::copy_if(begin(v), end(v),
    std::back_inserter(temp), odd);

std::vector<int> r;
std::transform(begin(temp), end(temp),
    std::back_inserter(r), square);
```

```
// C++ std containers and algorithms:
auto odd    = [](int n) { return n%2; };
auto square = [](int n) { return n*n; };
std::vector v = { 0,1,2,3,4,5,6,7,8,9 };

std::vector<int> temp;
std::copy_if(begin(v), end(v),
    std::back_inserter(temp), odd);

std::vector<int> r;
std::transform(begin(temp), end(temp),
    std::back_inserter(r), square);
```

issues:

- std algorithms lack composability
- need for containers of temporary values
- boilerplate code: pair of iterators (begin, end)

```
// C++20 ranges:
auto odd    = [](int n) { return n%2; };
auto square = [](int n) { return n*n; };

using namespace std::ranges::view;
auto r = iota(0)
    | filter(odd) | transform(square) | take(5);
```

```
// C++20 ranges:
auto odd    = [](int n) { return n%2; };
auto square = [](int n) { return n*n; };

using namespace std::ranges::view;
auto r = iota(0)
    | filter(odd) | transform(square) | take(5);
```

benefits:

- reducing boilerplate, improved readability, clear intent
- piping data through small computation bricks (view adaptors)
- lazy evaluation
  - no calculations up to here
  - data will be evaluated one at a time, when needed
  - optimized by compiler as efficient as hand-written loop

Data processing in other languages

- Unix pipes

```
cat text | grep 'C++' | sort | uniq | less
```

Data processing in other languages

- Unix pipes

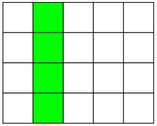
```
cat text | grep 'C++' | sort | uniq | less
```

- C# LINQ (lanquage integrated query)
  connects to databases, data sequences, . . .

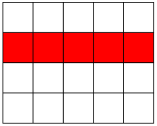
```
int[] numbers = new int[10]{ 0,1,2,3,4,5,6,7,8,9 };

var query =
  from num in numbers
  where (num%2) != 0
  select num*num;

foreach(int n in query) ...
```

SQL DB operations

```
SELECT name
```


projection

```
FROM table
WHERE age * 2 < 42;
```


selection

```
age * 2
```

$x \mapsto f(x)$ transformation

Background

C++98 standard library (aka “STL”)

containers

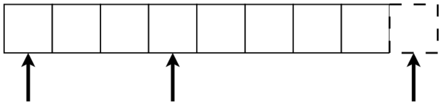
- sequential | associative | unordered (hashed)
- decoupled from algorithms by

iterators

- iterators for input and output sequences, e.g. `istream_iterator<int>`

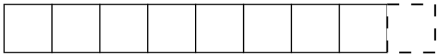
algorithms work on

- sequence with a
- range of
- iterators



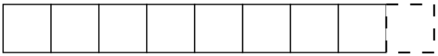
sequence

- (finite?) series of data values



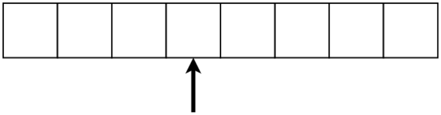
sequence

- (finite?) series of data values



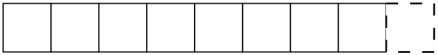
iterator

- mimic pointer to array
- refer to a value `*i`
- walk over values `++i`



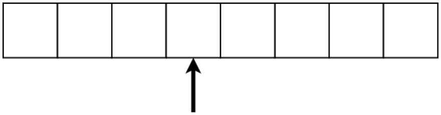
sequence

- (finite?) series of data values



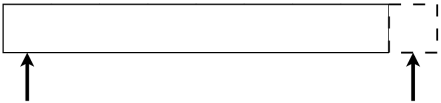
iterator

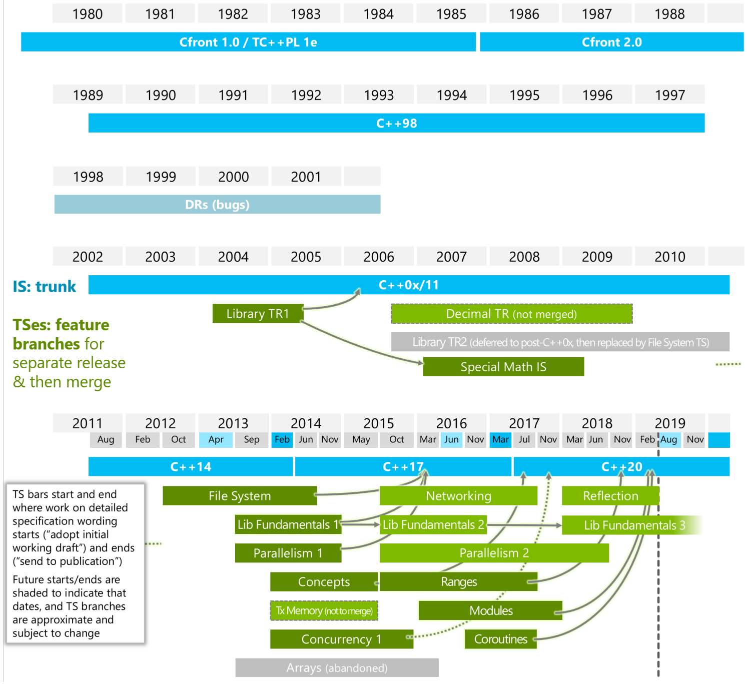
- mimic pointer to array
- refer to a value `*i`
- walk over values `++i`



range

- pair of iterators
  - `begin(r)`
  - `end(r)`
- half-open: one past the end
- may be empty:
  `begin(r) == end(r)`





TS bars start and end where work on detailed specification wording starts ("adopt initial working draft") and ends ("send to publication")

Future starts/ends are shaded to indicate that dates, and TS branches are approximate and subject to change

[isocpp.org]

History and ISO WG21 standardization process

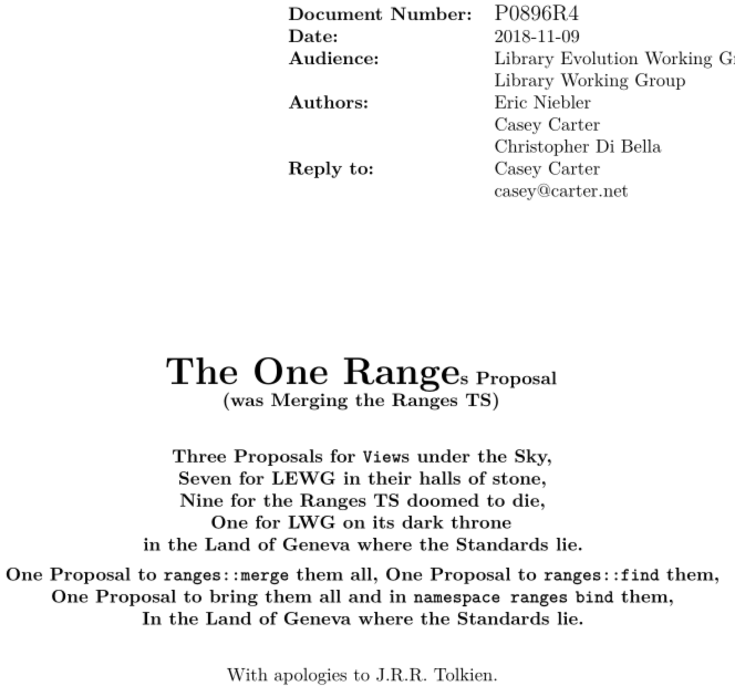
- Boost.Range ~ 2003
- range-v3 library (Eric Niebler) ~ Nov 2013
  - range adaptors, pipes
- Ranges TS, based on Concepts TS
  - concepts for ranges
  - rangified algorithms
- cmcstl2 (Casey M. Carter)
  - implements ranges as expected/adopted to C++20
- adopted to C++20 standard draft
  - P0789R3 : Range adaptors and utilities
  - P0896R4 : The One Ranges Proposal

History and ISO WG21 standardization process

- Boost.Range ~ 2003
- range-v3 library (Eric Niebler) ~ Nov 2013
  - range adaptors, pipes
- Ranges TS, based on Concepts TS
  - concepts for ranges
  - rangified algorithms
- cmcstl2 (Casey M. Carter)
  - implements ranges as expected/adopted to C++20
- adopted to C++20 standard draft
  - P0789R3 : Range adaptors and utilities
  - P0896R4 : The One Ranges Proposal

other major features with priority:

- Modules!
- Coroutines!
- Networking in C++23 or later?



Document Number: P0896R4
Date: 2018-11-09
Audience: Library Evolution Working Group, Library Working Group
Authors: Eric Niebler
Casey Carter
Christopher Di Bella
Reply to: Casey Carter
casey@carter.net

The One Ranges Proposal

(was Merging the Ranges TS)

Three Proposals for `Views` under the Sky,
Seven for LEWG in their halls of stone,
Nine for the Ranges TS doomed to die,
One for LWG on its dark throne
in the Land of Geneva where the Standards lie.

One Proposal to `ranges::merge` them all, One Proposal to `ranges::find` them,
One Proposal to bring them all and in `namespace ranges` bind them,
In the Land of Geneva where the Standards lie.

With apologies to J.R.R. Tolkien.

C++20 `<ranges>`

Ranges TS

- range concepts
  - constraining templates
- fundamental ranges
- rangified algorithms

and beyond

- range adaptors
- (some) view adaptors
- pipelining via `operator|`
- no action adaptors :-(

- Range
- OutputRange
- InputRange
- ForwardRange
- BidirectionalRange
- RandomAccessRange
- ContiguousRange
- CommonRange
- SizedRange
- ViewableRange
- View

Rangified algorithms

```
// C++17
std::sort(begin(v), end(v));
std::transform(
 begin(v), end(v),
 std::back_inserter(r),
 square);
```

```
// C++20
std::ranges::sort(v);
std::ranges::transform(
 v,
 std::ranges::back_inserter(r),
 square);
```

Rangified algorithms

```
// C++17
std::sort(begin(v), end(v));
std::transform(
 begin(v), end(v),
 std::back_inserter(r),
 square);
```

```
// C++20
std::ranges::sort(v);
std::ranges::transform(
 v,
 std::ranges::back_inserter(r),
 square);
```

- clearer error messages (e.g. `sort(list)`)
- more orthogonal interfaces through concepts

```
std::ranges::sort(range, compare, projection);
std::ranges::sort(first, sentinel, compare, projection);
    // compare = less{}
    // projection = identity{}
```

```cpp
struct Person
{
    std::string name;
    int born;
    int died;
};

auto lifespan = [](Person p) { return p.died - p.born; };

auto v = std::vector<Person>
{
    { "Galilei", 1564, 1642 },
    { "Newton" , 1642, 1726 },
    { "Hawking", 1942, 2018 },
};

rg::sort(v, {}, &Person::name);
rg::sort(v, {}, &Person::born);
rg::sort(v, rg::greater{}, lifespan);
```

Ranges

- container – O(N) copy
- view – O(1) copy
  - pair of iterators
  - iterator and sentinel (different types)
  - iterator and count (sized range)
- range-based for loop was redefined

```
for (auto e : r) ...
/// ==> C++17
{
  auto first = begin(r);
  auto last = end(r);     // allows different types
  for (; first != last; ++first)
  {
     auto e = *first;
     ...
  }
}
```

range access

- `[c][r]begin(r)`
- `[c][r]end(r)`
- `size(r)`
- `empty(r)`
- `[c]data(r)`

depending on range concept

range access

- `[c][r]begin(r)`
- `[c][r]end(r)`
- `size(r)`
- `empty(r)`
- `[c]data(r)`

depending on range concept

range adaptor [closure] objects

- accept a ViewableRange R
- return a View
- adaptor chain pipelines

```
C(R)
R | C
R | C | D
R | (C | D)
```

```
adaptor(range, args...)
adaptor(args...)(range)
range | adaptor(args...)
```

View adaptors so far (N4800, San Diego, Nov 2018)

in `namespace std::view = std::ranges::view`

`iota(start[, stop])`	integers from `start` (up to `stop`-1)
`take(n)`	first `n` elements
`filter(predicate)`	elements `e` which fulfill `pred(e)`
`transform(func)`	range of `e` $\mapsto$ range of `func(e)`
`reverse`	from back to front
`split(separators)`	split range into range of ranges
`join`	range of ranges into flat range
`all(r)`	
`empty<T>()`	range without elements of type T
`single(args...)`	one element range made from arguments
`counted(iter, n)`	iterates over `n` elements
`common(r)`	convert range iterators to same type

Examples

```
empty<int>()                      : []
single(42)                        : [42]
iota(0,6)                         : [0,1,2,3,4,5]
iota(0) | take(3)                 : [0,1,2]
all(v)                            : [0,1,2,3,4,5]
counted(begin(v),3)               : [0,1,2]
reverse(v)                        : [5,4,3,2,1,0]
v | reverse                       : [5,4,3,2,1,0]
filter(odd)                       : [1,3,5]
transform(square)                 : [0,1,4,9,16,25]

s                                 : AB CD E
s | split(' ')                    : [[A,B],[C,D],[E]]
s | split(' ') | join             : [A,B,C,D,E]
```

Infinite ranges

```
iota(0) | reverse; // compilation error!
iota(0) | take(10) | reverse; // o.k.
```

- you can't reverse an infinite range
  - obvious to humans
  - rejected by compiler due to concept checking

A frequently asked question

How to generate an index in a range-based for loop?

A frequently asked question

How to generate an index in a range-based for loop?

```
// C++11
{
  int i = 0;
  for (auto e : r)
  {
    ...
    ++i;
  }
}
```

```
// C++20
for (auto i = 0; auto e : r)
{
  ...
  ++i;
}
```

A frequently asked question

How to generate an index in a range-based for loop?

```
// C++11
{
  int i = 0;
  for (auto e : r)
  {
    ...
    ++i;
  }
}
```

```
// C++20
for (auto i = 0; auto e : r)
{
  ...
  ++i;
}
```

```
// C++17 with range-v3
for (auto [i, e] : enumerate(r)) ...
for (auto [i, e] : r | enumerate) ...
```

Not in C++20

```
// it could be so easy
auto enumerate = [](auto r) { return zip(iota(0), r); };
```

Not in C++20

```
// it could be so easy
auto enumerate = [](auto r) { return zip(iota(0), r); };
```

open questions about `zip(r1,r2)`:
pair? tuple? something else? proxy references?

Not in C++20

```
// it could be so easy
auto enumerate = [](auto r) { return zip(iota(0), r); };
```

open questions about `zip(r1,r2)`:
pair? tuple? something else? proxy references?

```
auto s = "ABC"s;
auto v = { 1, 2 };
```

```
// range-v3
s | enumerate           : [(0,A),(1,B),(2,C)]
cartesian_product(s,v): [(A,1),(A,2),(B,1),(B,2),(C,1),(C,2)]
```

Not in C++20 (yet?)

```
auto m = std::map{std::pair{'a', 1},{'b', 2},{'c', 3}};

// P1035R4:

m | keys                             : [a,b,c]
m | values                           : [1,2,3]
```

More action after C++20?

```
data
  | filter(pred)
  | transform(func)
  | action::sort(ordering, projection)
  | unique;
```

- `sort()` needs temporary container
  - collect all data processed up to here
  - therefore not a lazy view, called an action in range-v3

More action after C++20?

```
data
  | filter(pred)
  | transform(func)
  | action::sort(ordering, projection)
  | unique;
```

- `sort()` needs temporary container
  - collect all data processed up to here
  - therefore not a lazy view, called an action in range-v3

Proposals on the way:

- P1035R4 : Input range adaptors (`istream_view`)
  - straw polls on `take_while` `drop` `drop_while` `keys` `values`
- P1206R1 : `ranges::to<Container>`
- P1255R1 : `view::maybe`

range-v3 has more

. . . views

```
adjacent_filter adjacent_remove_if all any_view(rng)
bounded cartesian_product chunk common concat const_
counted cycle c_str delimit drop drop_exactly drop_while
empty enumerate filter for_each generate generate_n
group_by indirect intersperse iota join keys
linear_distribute move partial_sum remove_if repeat
repeat_n replace replace_if reverse sample single slice
sliding split stride tail take take_exactly take_while
tokenize transform unbounded unique values zip zip_with
```

. . . actions

```
drop drop_while erase insert join push_back push_front
remove_if shuffle slice sort split stable_sort stride take
take_while transform unique
```

Phantastic range libraries and where to find them

range-v3 master v0.40

```
#include <range/v3/all.hpp>
namespace rg = ranges;
```

range-v3 v1.0-beta (incomplete)

```
#include <range/v3/all.hpp>
namespace rg = ranges::cpp20; // C++20 adopted stuff only
```

cmcstl2

```
#include <experimental/ranges>
namespace rg = std::experimental::ranges;
```

C++20

```
#include <ranges>
namespace rg = std::ranges;
```

User experience

range-v3

- requires C++11
- uses concepts in gcc, if available (`g++ -fconcepts`)
    - (example failed 2019-03-06 locally & on wandbox.org)
- works with newest Visual C++
    - MSVC had a library fork for older compilers

cmcstl2

- install library in gcc experimental header path
- compile with `g++ -fconcepts` (GCC $\geq$ 6)
- found 2 bugs in implementation (were promptly fixed)
- too early to use in production code?

Example: range to container

```
// cmcstl2
auto r = iota(0)
    | filter(odd)
    | transform(square)
    | take(10)
//  | rg::to<std::vector> // range-v3, P1206
    ;
auto v = std::vector<int>{};
rg::copy(r, rg::back_inserter(v)); // not std::backinserter!
```

Example: range to container

```
// cmcstl2
auto r = iota(0)
    | filter(odd)
    | transform(square)
    | take(10)
//  | rg::to<std::vector> // range-v3, P1206
    ;
auto v = std::vector<int>{};
rg::copy(r, rg::back_inserter(v)); // not std::backinserter!

//  std::vector<int> v2 = r; // doesn't work
auto r2 = r | common;
auto v2 = std::vector<int>(begin(r2), end(r2));
```

Links and extra material

- https://isocpp.org
- https://wg21.link/NXXXX
  - N4800 C++20 working draft, San Diego
  - N4685 Ranges TS draft
- https://wg21.link/PXXXX
  - P0789 Range adaptors and utilities
  - P0896 The One Ranges Proposal
  - P1035 Input Range adaptors
  - P1206 ranges::to
  - P1255 view::maybe
- Eric Niebler's calendar demo
  - Github & Youtube
  - CppCon 2015 & C++Now 2015: Ranges for the Standard Library
- Bryce A. Lelbach: 2019-02 Kona ISO C++ Committee Trip Report. reddit.com

Libraries

- Github Eric Niebler range-v3
- Github Casey Carter cmcstl2

Compiler: MinGW GCC 8.1 / 8.2

- https://nuwen.net
- https://sourceforge.net/projects/mingw-w64/