AN OVERVIEW OF STANDARD RANGES

CppCon 2019

Tristan Brindle

Bryce Lelbach
@blelbach

Are you ready for @CppCon 2019?

cppcon bingo

Herb Sutter playing piano	Gripes about exceptions	Allocators	Monday WiFi issues	Unicode printing errors on badges
Memes on slides	Another hipster presentation uses reveal.js	Strategies for talking to C programmers	Attendees try to file feature requests in person	Template meta programming
Visual Studio demos	Zero cost abstractions	Boost	Concepts	Live coding demo crashes or doesn't compile
Bryce with a flock of volunteers following him	Assurances that X will be in the next standard	A lunch group grows way too big for any one restaurant	Java hate	Subtle bugs on concurrency slides
Last minute slide making	Cherry Coke	Monads	"JS/Swift/Rust has X, why doesn't C++?"	(Re)definition of modern C++

cppcon bingo

Herb Sutter playing piano	Gripes about exceptions	Allocators	Monday WiFi issues	Unicode printing errors on badges
Memes on slides	~~Another hipster presentation uses reveal.js~~	Strategies for talking to C programmers	Attendees try to file feature requests in person	Template meta programming
Visual Studio demos	Zero cost abstractions	Boost	Concepts	Live coding demo crashes or doesn't compile
Bryce with a flock of volunteers following him	Assurances that X will be in the next standard	A lunch group grows way too big for any one restaurant	Java hate	Subtle bugs on concurrency slides
Last minute slide making	Cherry Coke	Monads	"JS/Swift/Rust has X, why doesn't C++?"	(Re)definition of modern C++

cppcon bingo

Herb Sutter playing piano	Gripes about exceptions	Allocators	Monday WiFi issues	Unicode printing errors on badges
Memes on slides	Another hipster presentation uses reveal.js	Strategies for talking to C programmers	Attendees try to file feature requests in person	Template meta programming
Visual Studio demos	Zero cost abstractions	Boost	Concepts	Live coding demo crashes or doesn't compile
Bryce with a flock of volunteers following him	Assurances that X will be in the next standard	A lunch group grows way too big for any one restaurant	Java hate	Subtle bugs on concurrency slides
Last minute slide making	Cherry Coke	Monads	"JS/Swift/Rust has X, why doesn't C++?"	(Re)definition of modern C++

WHO AM I?

- Independent contractor/trainer based in London
- UK National Body member
- Director of C++ London Uni, a non-profit offering free beginner C++ classes

TODAY'S TALK

TODAY'S TALK

1. What's this ranges stuff all about?

TODAY'S TALK

1. What's this ranges stuff all about?
2. What's in it for me?

TODAY'S TALK

1. What's this ranges stuff all about?
2. What's in it for me?
3. How can I use this stuff today?

WHAT'S THIS RANGES STUFF ALL ABOUT?

- "STL 2.0"

- "STL 2.0"
- Lots of new features...

- "STL 2.0"
- Lots of new features...
- ...but not quite 100% backwards-compatible

- "STL 2.0"
- Lots of new features…
- …but not quite 100% backwards-compatible
- Most new facilities are in namespace `std::ranges`

- "STL 2.0"
- Lots of new features...
- ...but not quite 100% backwards-compatible
- Most new facilities are in namespace `std::ranges`
- Old code using `std::` will work as it did before

- "STL 2.0"
- Lots of new features...
- ...but not quite 100% backwards-compatible
- Most new facilities are in namespace `std::ranges`
- Old code using `std::` will work as it did before
- Will be part of C++20

- "STL 2.0"
- Lots of new features...
- ...but not quite 100% backwards-compatible
- Most new facilities are in namespace `std::ranges`
- Old code using `std::` will work as it did before
- Will be part of C++20
- Three implementations you can use today

WHAT IS A RANGE, ANYWAY?

A *range* is object on which you can call `begin()` and `end()`...

...where `begin()` returns an *iterator*, which can be incremented until it is equal to the thing returned from `end()`...

...like `std::vector`, for example

Ranges don't *replace* iterators...

Ranges don't *replace* iterators...

...they *build on them*

Ranges formalise many of the notions already implicit in the existing STL

WHAT'S IN IT FOR ME?

CONCEPTS

Concepts are a new feature in C++20

Concepts are a new feature in C++20

- Previously the Concepts TS

Concepts are a new feature in C++20

- Previously the Concepts TS
- Available in GCC and MSVC

Concepts are a new feature in C++20

- Previously the Concepts TS
- Available in GCC and MSVC
- Clang implementation in progress

Concepts allow us to control the instantiation of templates by testing syntactic conditions.

Concepts allow us to control the instantiation of templates by testing syntactic conditions.

"SFINAE on steroids"

A `concept` is a compile-time predicate which is `true` if the given type(s) meet the requirements

```
template <typename T>
concept string_convertible = requires(const T& t) {
    { t.to_string() } -> std::convertible_to<std::string>
};
```

```
template <typename T>
    requires string_convertible<T>
auto convert_to_string(const T& t) {
    return t.to_string();
}
```

```
template <string_convertible T>
auto convert_to_string(const T& t) {
    return t.to_string();
}
```

```
void convert_to_string(string_convertible auto&& range) {
    return t.to_string();
}
```

C++20 provides many "low-level" concepts such as `std::same_as` and `std::constructible_from` which replace the use of type traits

These can be used as "building blocks" for defining your own concepts

C++20 also provides higher-level concepts such as `std::bidirectional_iterator` and `std::random_access_range`

CONSTRAINED ALGORITHMS

```cpp
std::list<int> list{3, 2, 1};

std::sort(list.begin(), list.end());
```

```
In file included from example.cpp:2:
In file included from include/nanorange.hpp:10:
In file included from include/nanorange/algorithm.hpp:11:
In file included from include/nanorange/algorithm/adjacent_find.h
In file included from include/nanorange/ranges.hpp:17:
In file included from include/nanorange/detail/ranges/access.hpp:
In file included from include/nanorange/detail/ranges/begin_end.h
In file included from /Applications/Xcode.app/Contents/Developer/
In file included from /Applications/Xcode.app/Contents/Developer/
/Applications/Xcode.app/Contents/Developer/Toolchains/XcodeDefaul
        difference_type __len = __last - __first;
                                ~~~~~~ ^ ~~~~~~~
```

```
/Applications/Xcode.app/Contents/Developer/Toolchains/XcodeDefaul
    _VSTD::sort(__first, __last, __less<typename iterator_traits<
               ^
example.cpp:11:10: note: in instantiation of function template sp
    std::sort(list.begin(), list.end());
         ^
/Applications/Xcode.app/Contents/Developer/Toolchains/XcodeDefaul
operator-(const reverse_iterator<_Iter1>& __x, const reverse_iter
^
/Applications/Xcode.app/Contents/Developer/Toolchains/XcodeDefaul
operator-(const move_iterator<_Iter1>& __x, const move_iterator<_
^
/Applications/Xcode.app/Contents/Developer/Toolchains/XcodeDefaul
operator-(const __wrap_iter<_Iter1>& __x, const __wrap_iter<_Iter
^
```

```
/Applications/Xcode.app/Contents/Developer/Toolchains/XcodeDefaul
streamoff operator-(const fpos<_StateT>& __x, const fpos<_StateT>
          ^
In file included from example.cpp:2:
In file included from include/nanorange.hpp:10:
In file included from include/nanorange/algorithm.hpp:11:
In file included from include/nanorange/algorithm/adjacent_find.h
In file included from include/nanorange/ranges.hpp:17:
In file included from include/nanorange/detail/ranges/access.hpp:
In file included from include/nanorange/detail/ranges/begin_end.h
In file included from /Applications/Xcode.app/Contents/Developer/
In file included from /Applications/Xcode.app/Contents/Developer/
/Applications/Xcode.app/Contents/Developer/Toolchains/XcodeDefaul
                    if (__i >= __j)
                        ~~~ ^  ~~~
/Applications/Xcode.app/Contents/Developer/Toolchains/XcodeDefaul
```

```
/Applications/Xcode.app/Contents/Developer/Toolchains/XcodeDefaul
operator>=(const reverse_iterator<_Iter1>& __x, const reverse_ite
^
/Applications/Xcode.app/Contents/Developer/Toolchains/XcodeDefaul
operator>=(const move_iterator<_Iter1>& __x, const move_iterator<
^
/Applications/Xcode.app/Contents/Developer/Toolchains/XcodeDefaul
operator>=(const __wrap_iter<_Iter1>& __x, const __wrap_iter<_Ite
^
/Applications/Xcode.app/Contents/Developer/Toolchains/XcodeDefaul
operator>=(const __wrap_iter<_Iter1>& __x, const __wrap_iter<_Ite
^
/Applications/Xcode.app/Contents/Developer/Toolchains/XcodeDefaul
operator>=(const tuple<_Tp...>& __x, const tuple<_Up...>& __y)
^
/Applications/Xcode.app/Contents/Developer/Toolchains/XcodeDefaul
```

```
/Applications/Xcode.app/Contents/Developer/Toolchains/XcodeDefaul
operator>=(const reverse_iterator<_Iter1>& __x, const reverse_ite
^
/Applications/Xcode.app/Contents/Developer/Toolchains/XcodeDefaul
operator>=(const move_iterator<_Iter1>& __x, const move_iterator<
^
/Applications/Xcode.app/Contents/Developer/Toolchains/XcodeDefaul
operator>=(const __wrap_iter<_Iter1>& __x, const __wrap_iter<_Ite
^
/Applications/Xcode.app/Contents/Developer/Toolchains/XcodeDefaul
operator>=(const __wrap_iter<_Iter1>& __x, const __wrap_iter<_Ite
^
/Applications/Xcode.app/Contents/Developer/Toolchains/XcodeDefaul
operator>=(const tuple<_Tp...>& __x, const tuple<_Up...>& __y)
^
/Applications/Xcode.app/Contents/Developer/Toolchains/XcodeDefaul
```

442 lines of error messages!

```cpp
std::list<int> list{3, 2, 1};

ranges::sort(list.begin(), list.end());
```

```
example.cpp:13:5: error: no matching function for call to object
    ranges::sort(list.begin(), list.end());
    ^~~~~~~~~~~~
include/nanorange/algorithm/sort.hpp:20:5: note: candidate templa
    operator()(I first, S last, Comp comp = Comp{}, Proj proj = P
    ^
include/nanorange/algorithm/sort.hpp:31:5: note: candidate templa
    operator()(Rng&& rng, Comp comp = Comp{}, Proj proj = Proj{})
    ^
1 error generated.
```

RANGE-BASED OVERLOADS

```cpp
std::vector<int> vec{3, 2, 1};

std::sort(vec.begin(), vec.end());
```

```cpp
std::vector<int> vec{3, 2, 1};

std::ranges::sort(vec);
```

```cpp
std::vector<int> vec{3, 2, 1};

std::ranges::sort(vec);
```

At last! 🎉

Note of sadness: only the algorithms in `<algorithm>` will get range-based overloads in C++20

Note of sadness: only the algorithms in `<algorithm>` will get range-based overloads in C++20

The "other" algorithms in `<numeric>` will have to wait until C++23 😟

SENTINELS

In the existing STL, `end()` must return an iterator

In the ranges world, `end()` may return a *sentinel*

A sentinel is some type that is `equality_comparable_with` its corresponding iterator, which denotes the end of the range

Using a separate sentinel type allows us to simplify the definition of some iterators, and in some cases allows better codegen

```cpp
const std::string big_string = read_file();
// guaranteed to contain '\n'
```

```cpp
auto get_newline_pos(const std::string& str)
{
    return std::find(str.begin(), str.end(), '\n');
}
```

```
template <typename I, typename Val>
I find(I first, I last, const Val& val)
{
    while (first != last) {
        if (*first == val) {
            break;
        }
        ++first;
    }

    return first;
}
```

```
auto get_newline_pos(const std::string& str)
{
    return ranges::find(str.begin(),
                        ranges::unreachable_sentinel,
                        '\n');
}
```

```
template <typename I, typename S, typename Val>
I find(I first, unreachable_sentinel_t last, const Val& val)
{
    while (first != last) {
        if (*first == val) {
            break;
        }
        ++first;
    }

    return first;
}
```

```
template <typename I, typename Val>
I find(I first, unreachable_sentinel_t last, const Val& val)
{
    while (true) {
        if (*first == val) {
            break;
        }
        ++first;
    }

    return first;
}
```

https://godbolt.org/z/h3wFst

PROJECTIONS

- A *projection* is a *unary callable* which may be passed to most algorithms

- A *projection* is a *unary callable* which may be passed to most algorithms
- Projections modify the *view of the data* that the algorithm sees

```
struct Employee {
    std::string name;
    int id;
};

struct Payslip {
    std::string pay_info;
    int employee_id;
};

std::vector<Employee> employees;
std::vector<Payslip> payslips;
```

```
std::sort(employees.begin(), employees.end(),
    [] (const Employee& x, const Employee& y) {
        return x.id < y.id; });
)

std::sort(payslips.begin(), payslips.end(),
    [] (const Payslip& x, const Payslip& y) {
        return x.employee_id < y.employee_id; });

std::equal(employees.begin(), employees.end(),
    payslips.begin(), payslips.end(),
    [] (const Employee& e, const Payslip& p) {
        return e.id == p.employee.id; });
```

```
std::ranges::sort(employees,
    [] (const Employee& x, const Employee& y) {
        return x.id < y.id; });

std::ranges::sort(payslips,
    [] (const Payslip& x, const Payslip& y) {
        return x.employee_id < y.employee_id; });

std::ranges::equal(employees, payslips,
    [] (const Employee& e, const Payslip& p) {
        return e.id == p.employee_id; });
```

- A *projection* is a *unary callable* which may be passed to most algorithms

- A *projection* is a *unary callable* which may be passed to most algorithms
- Projections modify the *view of the data* that the algorithm sees

```
std::ranges::sort(employees,
    [] (const Employee& x, const Employee& y) {
        return x.id < y.id; });
```

```
std::ranges::sort(employees, std::ranges::less{},
    [] (const Employee& e) { return e.id; });
```

```
std::ranges::sort(employees, std::ranges::less{},
    [] (const Employee& e) { return e.id; });

std::ranges::sort(payslips, std::ranges::less{},
    [](const Payslip& p) { return p.employee_id; });

std::ranges::equal(employees, payslips,
    std::ranges::equal_to{},
    [] (const Employee& e) { return e.id; },
    [] (const Payslip& p) { return p.employee_id; });
```

```
std::ranges::sort(employees, std::ranges::less{},
    &Employee::id);

std::ranges::sort(payslips, std::ranges::less{},
    &Payslip::employee_id);

std::ranges::equal(employees, payslips,
    std::ranges::equal_to{},
    &Employee::id, &Payslip::employee_id);
```

```
std::ranges::sort(employees, {}, &Employee::id);

std::ranges::sort(payslips, {}, &Payslip::employee_id);

std::ranges::equal(employees, payslips, {},
    &Employee::id, &Payslips::employee_id);
```

```
std::sort(employees.begin(), employees.end(),
    [] (const Employee& x, const Employee& y) {
        return x.id < y.id; });
)

std::sort(payslips.begin(), payslips.end(),
    [] (const Payslip& x, const Payslip& y) {
        return x.employee_id < y.employee_id; });

std::equal(employees.begin(), employees.end(),
    payslips.begin(), payslips.end(),
    [] (const Employee& e, const Payslip& p) {
        return e.id == p.employee.id; });
```

```
std::ranges::sort(employees, {}, &Employee::id);

std::ranges::sort(payslips, {}, &Payslip::employee_id);

std::ranges::equal(employees, payslips, {},
    &Employee::id, &Payslips::employee_id);
```

VIEWS

- The standard algorithms are great!

- The standard algorithms are great!
- But they don't *compose* well

- The standard algorithms are great!
- But they don't *compose* well
- They perform their operations *eagerly*

```
void print_squares(const vector<int>& vec)
{
    for (int i : vec) {
        cout << i * i;
    }
}
```

"No raw loops!"

- Sean Parent, "C++ Seasoning"

```
void print_squares(const vector<int>& vec)
{
    ranges::transform(vec, ostream_iterator<int>{cout},
        [] (int i) {
            return i * i;
        }
    );
}
```

```
void print_even_squares(const vector<int>& vec)
{
    for (int i : vec) {
        if (i % 2 == 0) {
            cout << i * i;
        }
    }
}
```

```
void print_even_squares(vector<int> vec)
{
    auto removed = ranges::remove_if(vec, [] (int i) {
        return i % 2 != 0
    });
    ranges::transform(vec.begin(), removed.begin(),
        ostream_iterator<int>{cout},
        [] (int i) {
            return i * i;
        });
}
```

C++20 will include new *range adaptors* ("views") which offer *lazy evaluation* instead

```cpp
void print_even_squares(const std::vector<int>& vec)
{
    auto square = [](auto i) { return i * i; };
    auto is_even = [](auto i) { return i % 2 == 0; };

    auto view = ranges::views::transform(
        ranges::views::filter(vec, is_even),
        square);

    ranges::copy(view, ostream_iterator<int>{cout});
}
```

```cpp
void print_even_squares(const std::vector<int>& vec)
{
    auto square = [](auto i) { return i * i; };
    auto is_even = [](auto i) { return i % 2 == 0; };

    auto view = vec
        | ranges::view::filter(is_even)
        | ranges::view::transform(square);

    ranges::copy(view, ostream_iterator<int>{cout});
}
```

From https://github.com/tcbrindle/utf_ranges

```
void utf8_to_utf16be(std::istream& in_file, std::ostream& out_fil
    auto view = utf::istreambuf(in_file)
            // Remove UTF-8 "BOM" if present
            | utf::view::consume_bom
            // Convert to UTF-16
            | utf::view::utf16
            // Prepend UTF-16 BOM to start of range
            | utf::view::add_bom
            // Convert to big-endian
            | utf::view::endian_convert<endian::order::big>
            // Write out as bytes
            | utf::view::bytes;

    // Do the copy
    rng::copy(view, utf::ostreambuf_iterator<char>{out_file});
```

HOW CAN I USE THIS STUFF TODAY?

Range-V3 https://github.com/ericniebler/range-v3

Range-V3 https://github.com/ericniebler/range-v3

- Eric Niebler's original ranges implementation

Range-V3 https://github.com/ericniebler/range-v3

- Eric Niebler's original ranges implementation
- Has **many** extra views and actions which are not part of C++20

Range-V3 https://github.com/ericniebler/range-v3

- Eric Niebler's original ranges implementation
- Has **many** extra views and actions which are not part of C++20
- Very popular, widely used

Range-V3 https://github.com/ericniebler/range-v3

- Eric Niebler's original ranges implementation
- Has **many** extra views and actions which are not part of C++20
- Very popular, widely used
- Uses C++14, works with all major compilers

Range-V3 https://github.com/ericniebler/range-v3

- Eric Niebler's original ranges implementation
- Has **many** extra views and actions which are not part of C++20
- Very popular, widely used
- Uses C++14, works with all major compilers
- Will use language concepts if available

CMCSTL2 https://github.com/CaseyCarter/CMCSTL2

CMCSTL2 https://github.com/CaseyCarter/CMCSTL2

- Casey Carter's reference implementation of ranges

CMCSTL2 https://github.com/CaseyCarter/CMCSTL2

- Casey Carter's reference implementation of ranges
- Uses language concepts only

CMCSTL2 https://github.com/CaseyCarter/CMCSTL2

- Casey Carter's reference implementation of ranges
- Uses language concepts only
- ...so no Clang ~~or MSVC~~ support yet

CMCSTL2 https://github.com/CaseyCarter/CMCSTL2

- Casey Carter's reference implementation of ranges
- Uses language concepts only
- ...so no Clang ~~or MSVC~~ support yet
- A couple of extensions that are not part of the proposals

NanoRange https://github.com/tcbrindle/NanoRange

NanoRange https://github.com/tcbrindle/NanoRange

- My implementation of the ranges specifications

NanoRange https://github.com/tcbrindle/NanoRange

- My implementation of the ranges specifications
- Uses C++17, works with all major compilers

NanoRange https://github.com/tcbrindle/NanoRange

- My implementation of the ranges specifications
- Uses C++17, works with all major compilers
- Uses ~~hideous~~ template magic to emulate concepts

NanoRange https://github.com/tcbrindle/NanoRange

- My implementation of the ranges specifications
- Uses C++17, works with all major compilers
- Uses ~~hideous~~ template magic to emulate concepts
- No extensions, just the proposed features

NanoRange https://github.com/tcbrindle/NanoRange

- My implementation of the ranges specifications
- Uses C++17, works with all major compilers
- Uses ~~hideous~~ template magic to emulate concepts
- No extensions, just the proposed features
- Aims to provide a smooth upgrade path to std::ranges

THANK YOU VERY MUCH!

QUESTIONS?

AN OVERVIEW OF STANDARD RANGES

Twitter: @tristanbrindle

NanoRange: github.com/tcbrindle/NanoRange