



What Is a Range?

Šimon Tóth

2024

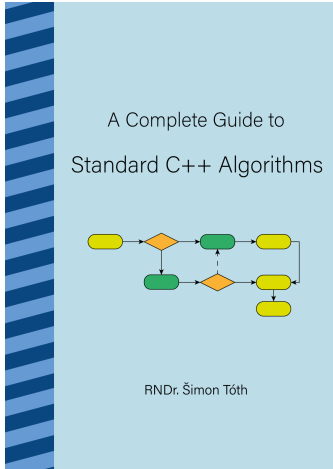
Slides

<https://github.com/HappyCerberus/what-is-a-range>



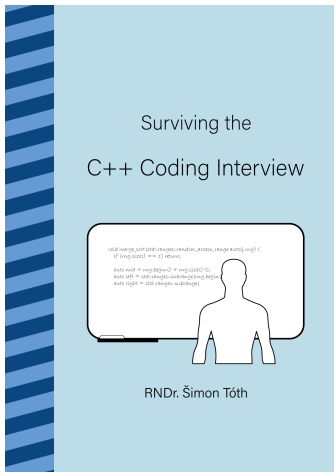
Ask questions!

A Complete Guide to Standard C++ Algorithms




- ▶ Content complete for C++20
- ▶ Free on GitHub:
[HappyCerberus/book-cpp-algorithms](https://github.com/HappyCerberus/book-cpp-algorithms)
- ▶ Donate to EFF on LeanPub:
leanpub.com/cpp-algorithms-guide

Surviving the C++ Coding Interview



- ▶ Work in progress (95 pages)
- ▶ Free Community Edition:
leanpub.com/cpp-coding-interview/signup
- ▶ Donate to EFF on LeanPub:
leanpub.com/cpp-coding-interview

Daily bit(e) of C++

**Simon Toth** ***
C++ Educational Content Creator | 20 years of Software Engineering experience distilled into dige...
3d

A default-constructed `std::variant` will always initialize with its first element as active.

This can be problematic because it requires the value construction of the first element, which can be costly, or the element may not even support value construction.

To avoid this issue and allow a semantically "empty" `std::variant`, we can use the special type `std::monostate` as the first argument of the `std::variant`.

Compiler Explorer link: <https://lnkd.in/eSaY9ne5>

#cpp #cplusplus #coding #programming #dailybiteofcpp

```
#include <variant>

struct X {
    X(int v) : v_(v) {}
    int v_;
};

struct Y {
    Y(double v) : v_(v) {}
    double v_;
};


// X, Y are non-default constructible
// std::variant<X,Y> a; wouldn't compile, can't default construct

std::variant<X,Y> a = Y(20); // OK
// a.index() == 1

std::variant<std::monostate,X,Y> b; // OK, semantically "empty"
// b.index() == 0, std::holds_alternative<std::monostate>(b) == true

std::variant<int,double> c;
std::variant<int,double> d(0);
// c == d

// to distinguish an empty state:
std::variant<std::monostate,int,double> e;
std::variant<std::monostate,int,double> f(0);
// e != f
```

 62 · 3 Comments

- ▶ [linkedin.com/in/simontoth](https://www.linkedin.com/in/simontoth)
- ▶ hachyderm.io/@simontoth
- ▶ medium.com/@simontoth
- ▶ simontoth.substack.com
- ▶ github.com/HappyCerberus/daily-bite-cpp

What is a range? (pre-C++20)

```
auto first = container.begin();  
auto last = container.end();
```



```
auto first = container.begin();  
auto last = container.end();  
  
using std::begin;  
auto first = begin(container);  
using std::end;  
auto last = end(container);
```

```
struct Member {  
    struct Iterator {};  
    Iterator begin() { return {}; }  
};
```

```
using std::begin;
```

```
Member m;
```

```
auto it = begin(m);
```

```
// decltype(it) == Member::Iterator
```

<https://compiler-explorer.com/z/qnTxYcr7z>

```
int arr[5];
```

```
using std::begin;
```

```
auto it = begin(arr);
```

```
// it == arr
```

```
// decltype(it) == int*
```

<https://compiler-explorer.com/z/qnTxYcr7z>

```
struct Friend {  
    struct Iterator {};  
    friend Iterator begin(const Friend&) { return {}; }  
};
```

```
using std::begin;
```

```
Friend f;
```

```
auto it = begin(f);
```

```
// decltype(it) == Friend::Iterator
```

<https://compiler-explorer.com/z/qnTxYcr7z>

```
auto first = container.begin();  
auto last = container.end();  
  
for (auto it = first; it != last; ++it)  
    std::println("{} ", *it);
```

begin

end

begin

end

|
begin

|
mid

|
end

begin

mid

mid

end

[first, last)

$\{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}$
 $[0, 10)$

```
int data[5] = {1,2,3,4,5};
```

<https://compiler-explorer.com/z/89d5f6h7b>

```
int data[5] = {1,2,3,4,5};
```

```
int* first = data;
```

```
int* last = first + 5; // one past
```

<https://compiler-explorer.com/z/89d5f6h7b>

```
int data[5] = {1,2,3,4,5};  
  
int* first = data;  
int* last = first + 5; // one past  
  
for (int* it = first; it != last; ++it)  
    std::println("{} ", *it);
```

<https://compiler-explorer.com/z/89d5f6h7b>

```
List list = {1,2,3,4,5};
```

```
Node* first = list.head();
```

```
Node* last = nullptr;
```

```
for (Node* it = first; it != last; it = it->next)  
    std::println("{} ", it->value);
```

<https://compiler-explorer.com/z/5Ecdhqa4>

```
for (int* it = first; it != last; ++it)  
    std::println("{} ", *it);
```

```
for (Node* it = first; it != last; it = it->next)  
    std::println("{} ", it->value);
```



```
std::forward_list<int> list{1,2,3,4,5};  
  
auto first = list.begin();  
auto last = list.end();  
  
for (auto it = first; it != last; ++it)  
    std::println("{} ", *it);
```

<https://compiler-explorer.com/z/jrP9GeG5e>

input/output

- ▶ `std::istream_iterator, std::ostream_iterator`
- ▶ `base: *it, ++it, it++`

forward

- ▶ `std::forward_list`
- ▶ `base: *it, ++it, it++`
- ▶ `multi-access: auto it2 = it1; ++it1; *it1; *it2;`

bidirectional

- ▶ `std::set, std::map`
- ▶ `base: *it, ++it, it++`
- ▶ `multi-access: auto it2 = it1; ++it1; *it1; *it2;`
- ▶ `bidirectional: --it, it--`

random access

- ▶ `std::deque`
- ▶ base: `*it, ++it, it++`
- ▶ multi-access: `auto it2 = it1; ++it1; *it1; *it2;`
- ▶ bidirectional: `--it, it--`
- ▶ *random access*: `it + offset, it - offset`
- ▶ *distance*: `it1 - it2`

contiguous

- ▶ `std::array`, `std::vector`
- ▶ base: `*it`, `++it`, `it++`
- ▶ multi-access: `auto it2 = it1; ++it1; *it1; *it2;`
- ▶ bidirectional: `--it`, `it--`
- ▶ random access: `it + offset`, `it - offset`
- ▶ distance: `it1 - it2`
- ▶ *contiguous memory*

What is a range?

► `for (auto e : rng) {}`

- ▶ `for (auto e : rng) {}`
- ▶ `begin(rng), end(rng)`

- ▶ `for (auto e : rng) {}`
- ▶ `begin(rng), end(rng)`
- ▶ input or output iterator

- ▶ `for (auto e : rng) {}`
- ▶ `begin(rng), end(rng)`
- ▶ input or output iterator
- ▶ `[begin(rng), end(rng))`

```
namespace std::ranges {  
    template<class T>  
    concept range = requires(T& t) {  
        ranges::begin(t);  
        ranges::end    (t);  
    };  
}
```

```
int arr[5] = {1,2,3,4,5};

auto first = std::ranges::begin(arr);
auto last = std::ranges::end(arr);

for (auto it = first; it != last; ++it)
    std::println("{} ", *it);
```

<https://compiler-explorer.com/z/b67PnGWP8>

```
int arr[5] = {1,2,3,4,5};

auto first = std::ranges::begin(arr);
auto last = std::ranges::end(arr);

for (auto& e : std::ranges::subrange(first, last))
    std::println("{} ", e);
```

<https://compiler-explorer.com/z/dMozG7j8h>

Range concepts

- ▶ `ranges::input_range`
- ▶ `ranges::output_range`
- ▶ `ranges::forward_range`
- ▶ `ranges::bidirectional_range`
- ▶ `ranges::random_access_range`
- ▶ `ranges::contiguous_range`

```
auto sum(std::ranges::input_range auto&& rng) {  
    auto result = std::ranges::range_value_t<decltype(rng)>{};  
    for (auto& e : rng)  
        result += e;  
    return result;  
}
```

<https://compiler-explorer.com/z/f5h7jYnjP>


```
template <std::ranges::input_range Rng>
auto sum(Rng&& rng) {
    auto result = std::ranges::range_value_t<Rng>{};
    for (auto& e : rng)
        result += e;
    return result;
}
```

<https://compiler-explorer.com/z/a9qx5rWh1>

Universal reference

```
const char& find_second(const std::string& str, char c) {  
    static char not_found = '\\0';  
  
    size_t idx = str.find(c);  
    if (idx == std::string::npos) return not_found;  
  
    idx = str.find(c, idx+1);  
    if (idx == std::string::npos) return not_found;  
  
    return str[idx];  
}
```

<https://compiler-explorer.com/z/E6EqjzTKr>

```
const char& find_second(const std::string& str, char c) {  
    static char not_found = '\\0';  
  
    size_t idx = str.find(c);  
    if (idx == std::string::npos) return not_found;  
  
    idx = str.find(c, idx+1);  
    if (idx == std::string::npos) return not_found;  
  
    return str[idx];  
}
```

```
const char &c = find_second("Hello World!", 'o');
```

<https://compiler-explorer.com/z/E6EqjzTKr>

```
const char& find_second(const std::string_view& str, char c) {  
    static char not_found = '\\0';  
  
    size_t idx = str.find(c);  
    if (idx == std::string::npos) return not_found;  
  
    idx = str.find(c, idx+1);  
    if (idx == std::string::npos) return not_found;  
  
    return str[idx];  
}
```

```
const char &c = find_second("Hello World!", 'o');
```

<https://compiler-explorer.com/z/79Wej378n>

Borrowed range

- ▶ `std::string_view`
- ▶ `std::span`

Range algorithms

```
auto it1 = std::ranges::find(std::string_view("Hello World"), 'o');  
// decltype(it1) == std::string_view::iterator  
// *it1 == 'o'
```

Range algorithms

```
auto it1 = std::ranges::find(std::string_view("Hello World!"), 'o');  
// decltype(it1) == std::string_view::iterator  
// *it1 == 'o'
```

```
auto it2 = std::ranges::find(std::string("Hello World!"), 'o');  
// decltype(it2) == std::ranges::dangling
```


Range algorithms

```
auto it1 = std::ranges::find(std::string_view("Hello World!"), 'o');  
// decltype(it1) == std::string_view::iterator  
// *it1 == 'o'
```

```
auto it2 = std::ranges::find(std::string("Hello World!"), 'o');  
// decltype(it2) == std::ranges::dangling
```

```
std::string str1("Hello World!");  
auto it3 = std::ranges::find(str1, 'o');  
// decltype(it3) == std::string::iterator
```

```
std::string_view str2("Hello World!");  
auto it4 = std::ranges::find(str2, 'o');  
// decltype(it4) == std::string_view::iterator
```

<https://compiler-explorer.com/z/E5cMG4qqT>

```
void fun(const std::string& rng) {}  
  
fun(std::string("")); // taking ownership  
  
std::string str;  
fun(str); // borrowing
```

```
void fun(auto&& rng) {  
    if constexpr (std::ranges::borrowed_range<decltype(rng)>) {  
        // borrowing  
    } else {  
        // taking ownership  
    }  
}
```

```
fun(std::string(""));    // taking ownership  
fun(std::string_view("")); // borrowing
```

```
std::string str;  
fun(str);           // borrowing  
fun(std::as_const(str)); // borrowing
```

<https://compiler-explorer.com/z/h436jzTaq>

```
template <std::ranges::forward_range Str>
requires std::ranges::borrowed_range<Str>
    && std::same_as<std::ranges::range_value_t<Str>, char>
const char& find_second(Str&& str, char c) {
    /* ... */
}
```

<https://compiler-explorer.com/z/x68W8PGEq>

```
template <std::ranges::input_range Rng>
auto sum(Rng&& rng) {
    auto result = std::ranges::range_value_t<Rng>{};
    for (auto& e : rng)
        result += e;
    return result;
}
```

Views

C++ Views

Nicolai M. Josuttis

josuttis.com

[@NicoJosuttis](https://twitter.com/NicoJosuttis)

04/23

C++

©2023 by josuttis.com

1

josuttis | eckstein
IT communication

ACCU.ORG

46



Nico Josuttis

View

- ▶ cheap to move
- ▶ cheap to destroy when moved-from
- ▶ cheap to copy if copyable

Not a borrowed range

```
std::vector<int> data{1,2,3,4,5};  
auto rng1 = std::views::all(data);  
// borrowed, decltype(rng1) == std::ranges::ref_view<...>  
  
auto rng2 = std::views::all(std::vector<int>{1,2,3,4,5});  
// not borrowed, decltype(rng2) == std::ranges::owning_view<...>
```

<https://compiler-explorer.com/z/T74ffK1dz>

Views as code

```
std::vector<int> data{1,2,3,4,5};

auto view = data |
    std::views::transform([](int v) { return v * 2; }) |
    std::views::transform([](int v) { return v + 1; });
// view == {3, 5, 7, 9, 11}
```

<https://compiler-explorer.com/z/vWE9hq7Mq>

Views as code

```
std::vector<int> data{1,2,3,4,5};

auto view = data |
    std::views::transform([](int v) { return v * 2; }) |
    std::views::transform([](int v) { return v + 1; });
// view == {3, 5, 7, 9, 11}

auto manifested = view |
    std::ranges::to<std::vector>();
// decltype(manifested) == std::vector<int>
// manifested == {3, 5, 7, 9, 11}
```

<https://compiler-explorer.com/z/6Ejjz8b7Y>

Views as code

```
std::vector<int> data{1,2,3,4,5};  
  
auto view = data |  
    std::views::filter([](int v) { return v % 2 == 0; });  
// view == {2, 4}
```

<https://compiler-explorer.com/z/fbsPc9EYx>

Views as code

```
std::vector<int> data{1,3,5,7,9};

auto view = data |
    std::views::filter([](int v) { return v % 2 == 0; });
// view == {}

bool empty = view.begin() == view.end();
// empty == true
```

<https://compiler-explorer.com/z/j86jKrb5f>

Views as code

```
std::list<int> data{1,3,5,7,9};

auto view = data |
    std::views::filter([](int v) { return v % 2 == 0; });
// view == {}

bool empty = view.begin() == view.end();
// empty == true

data.push_front(2);

empty = view.begin() == view.end();
// empty == true
```

<https://compiler-explorer.com/z/WedM1eE8z>

Views as code

```
void fn(const auto& rng) {  
    auto b = rng.begin();  
}
```

```
fn(std::vector<int>{1,3,5,7,9} |  
    std::views::filter([](int v) { return v % 2 == 0; }));
```

<https://compiler-explorer.com/z/hzd5Ksezd>

Views as code

```
void fn(auto&& rng) {  
    auto b = rng.begin();  
}
```

```
fn(std::vector<int>{1,3,5,7,9} |  
    std::views::filter([](int v) { return v % 2 == 0; }));
```

<https://compiler-explorer.com/z/G1Tzf3ePa>

When taking a range as an argument, always use a universal reference.

Input ranges

`std::views::istream`

// standard input: 1 2 3 4 5 6 7 8 9

std::views::istream

// standard input: 1 2 3 4 5 6 7 8 9

```
std::vector<int> out1;  
std::ranges::copy(  
    std::views::istream<int>(std::cin) | std::views::take(3),  
    std::back_inserter(out1)  
);
```

```
std::vector<int> out2;  
std::ranges::copy(  
    std::views::istream<int>(std::cin) | std::views::take(3),  
    std::back_inserter(out2)  
);
```

std::views::istream

```
// standard input: 1 2 3 4 5 6 7 8 9
```

```
std::vector<int> out1;  
std::ranges::copy(  
    std::views::istream<int>(std::cin) | std::views::take(3),  
    std::back_inserter(out1)  
);
```

```
std::vector<int> out2;  
std::ranges::copy(  
    std::views::istream<int>(std::cin) | std::views::take(3),  
    std::back_inserter(out2)  
);
```

```
// out1 == {1, 2, 3}
```

```
// out2 == {5, 6, 7}
```

<https://compiler-explorer.com/z/YGvb57K6P>

it

end

it

end

std::views::istream

```
// standard input: 1 2 3 4 5 6 7 8 9
```

```
std::vector<int> out1;  
std::ranges::copy(  
    std::views::istream<int>(std::cin) | std::views::take(3),  
    std::back_inserter(out1)  
);
```

```
std::vector<int> out2;  
std::ranges::copy(  
    std::views::istream<int>(std::cin) | std::views::take(3),  
    std::back_inserter(out2)  
);
```

```
// out1 == {1, 2, 3}
```

```
// out2 == {5, 6, 7}
```

<https://compiler-explorer.com/z/YGvb57K6P>

std::views::istream

// standard input: 1 2 3 4 5 6 7 8 9

```
std::vector<int> out1;  
auto [in, out] = std::ranges::copy(  
    std::views::istream<int>(std::cin) | std::views::take(3),  
    std::back_inserter(out1)  
);
```

<https://compiler-explorer.com/z/8Kvjse9PY>

std::views::istream

```
// standard input: 1 2 3 4 5 6 7 8 9  
  
std::vector<int> out1;  
auto [in, out] = std::ranges::copy(  
    std::views::istream<int>(std::cin) | std::views::take(3),  
    std::back_inserter(out1)  
);  
// decltype(in) == std::ranges::dangling
```

<https://compiler-explorer.com/z/8Kvjse9PY>

std::views::istream

```
// standard input: 1 2 3 4 5 6 7 8 9  
  
auto view = std::views::istream<int>(std::cin) | std::views::take(3);  
  
std::vector<int> out1;  
auto [in, out] = std::ranges::copy(  
    view,  
    std::back_inserter(out1)  
);  
// out1 == {1,2,3}  
// *in == 4
```

<https://compiler-explorer.com/z/zPdjrvmhx>

Sized range

```
auto count_to_five = std::views::iota(1) |  
    std::views::take(5);
```

```
std::vector<int> store;
```

<https://compiler-explorer.com/z/138hEf7bj>

```
auto count_to_five = std::views::iota(1) |  
    std::views::take(5);
```

```
std::vector<int> store;
```

```
store.reserve(count_to_five.size());
```

<https://compiler-explorer.com/z/138hEf7bj>

```
auto count_to_five = std::views::iota(1) |  
    std::views::take(5);  
  
std::vector<int> store;  
  
store.reserve(count_to_five.size()); // Will not compile  
std::ranges::copy(count_to_five, std::back_inserter(store));  
// std::ranges::sized_range<decltype(count_to_five)> == false
```

<https://compiler-explorer.com/z/138hEf7bj>

```
auto count_to_five = std::views::iota(1) |  
    std::views::take(5);  
  
auto wrapped = std::ranges::subrange(count_to_five, 5);  
  
std::vector<int> store;  
  
store.reserve(wrapped.size());  
std::ranges::copy(wrapped, std::back_inserter(store));  
// std::ranges::sized_range<decltype(count_to_five)> == false  
// std::ranges::sized_range<decltype(wrapped)> == true
```

<https://compiler-explorer.com/z/77s55j7qY>

Common range

```
namespace std::ranges {  
    template<class T>  
    concept range = requires(T& t) {  
        ranges::begin(t);  
        ranges::end  (t);  
    };  
}
```

```
std::vector<int> data{1,2,3,4,5};  
// std::ranges::common_range<decltype(data)> == true
```

<https://compiler-explorer.com/z/86nocb3ra>

```
std::vector<int> data{1,2,3,4,5};  
// std::ranges::common_range<decltype(data)> == true  
  
auto first = data.begin();  
// decltype(first) == std::vector<int>::iterator  
auto last = data.end();  
// decltype(last) == std::vector<int>::iterator
```

<https://compiler-explorer.com/z/86nocb3ra>

```
std::vector<int> data{1,2,3,4,5};  
// std::ranges::common_range<decltype(data)> == true  
  
auto first = data.begin();  
// decltype(first) == std::vector<int>::iterator  
auto last = data.end();  
// decltype(last) == std::vector<int>::iterator  
  
int sum = std::accumulate(first, last, 0);  
// OK, sum == 15
```

<https://compiler-explorer.com/z/86nocb3ra>

```
auto iota = std::views::iota(1) |  
    std::views::take(5);  
// std::ranges::common_range<decltype(iota)> == false
```

<https://compiler-explorer.com/z/Yd3qj4515>

```
auto iota = std::views::iota(1) |  
    std::views::take(5);  
// std::ranges::common_range<decltype(iota)> == false  
  
auto first = iota.begin();  
auto last = iota.end();  
// decltype(first) != decltype(last)  
  
// Will not compile  
int sum = std::accumulate(first, last, 0);
```

<https://compiler-explorer.com/z/Yd3qj4515>

```
auto iota = std::views::iota(1) |  
    std::views::take(5) |  
    std::views::common;  
// std::ranges::common_range<decltype(iota)> == true
```

```
auto first = iota.begin();  
auto last = iota.end();  
// decltype(first) == decltype(last)
```

```
int sum = std::accumulate(first, last, 0);  
// OK, sum == 15
```

<https://compiler-explorer.com/z/7sqGdsPqP>


```
auto iota = std::views::iota(1) |  
    std::views::take(5);  
// std::ranges::common_range<decltype(iota)> == false  
  
int sum = std::ranges::fold_left(iota, 0, std::plus<>{});  
// OK, sum == 15
```

<https://compiler-explorer.com/z/7f3YTY9TT>

```
struct count_to_five {
    struct iterator {
        int v{1};
        iterator& operator++() { ++v; return *this; }
        int operator*() { return v; }
        bool operator==(const std::default_sentinel_t&) const {
            return v > 5;
        }
    };
    iterator begin() { return {}; }
    std::default_sentinel_t end() { return {}; }
};
```

```
for (auto v : count_to_five{}) {}  
// {1, 2, 3, 4, 5}
```

<https://compiler-explorer.com/z/fjzxq44cq>

```
std::string text = "first line\nsecond line\n";  
assert(text.back() == '\\n');  
  
auto delim = std::ranges::find(text, '\\n');  
auto line = std::ranges::subrange(text.begin(), delim);
```

<https://compiler-explorer.com/z/q9MYxPqv7>

```
std::string text = "first line\nsecond line\n";  
assert(text.back() == '\\n');
```

```
auto first = text.begin();  
auto last = text.end();  
auto delim = first;  
for (; delim != last; ++delim)  
    if (*delim == '\\n')  
        break;
```

```
auto line = std::ranges::subrange(text.begin(), delim);
```

<https://compiler-explorer.com/z/5GPs44adT>

```
std::string text = "first line\nsecond line\n";  
assert(text.back() == '\\n');  
  
auto first = text.begin();  
auto delim = first;  
for (; *delim != '\\n'; ++delim);  
  
auto line = std::ranges::subrange(text.begin(), delim);
```

<https://compiler-explorer.com/z/1M4cTsG9x>

```
std::string text = "first line\nsecond line\n";  
assert(text.back() == '\\n');  
  
auto unbounded = std::ranges::subrange(  
    text.begin(), std::unreachable_sentinel);  
  
auto delim = std::ranges::find(unbounded, '\\n');  
auto line = std::ranges::subrange(text.begin(), delim);
```

<https://compiler-explorer.com/z/h1eE4T168>

```
template <
    std::input_iterator It,
    std::sentinel_for<It> Sentinel>
void fn(It first, Sentinel last) {
    for (auto it = first; it != last; ++it)
        std::println("{} ", *it);
}
```

```
auto view = std::views::iota(1) |
    std::views::take(5);
```

```
fn(view.begin(), view.end());
```

<https://compiler-explorer.com/z/Yhna9eEd7>

Summary

- ▶ `ranges::input_range`, `ranges::output_range`, `ranges::forward_range`,
`ranges::bidirectional_range`, `ranges::random_access_range`,
`ranges::contiguous_range`

Summary

- ▶ `ranges::input_range`, `ranges::output_range`, `ranges::forward_range`,
`ranges::bidirectional_range`, `ranges::random_access_range`,
`ranges::contiguous_range`
- ▶ `ranges::borrowed_range`

Summary

- ▶ `ranges::input_range`, `ranges::output_range`, `ranges::forward_range`,
`ranges::bidirectional_range`, `ranges::random_access_range`,
`ranges::contiguous_range`
- ▶ `ranges::borrowed_range`
- ▶ `ranges::view`

Summary

- ▶ `ranges::input_range`, `ranges::output_range`, `ranges::forward_range`,
`ranges::bidirectional_range`, `ranges::random_access_range`,
`ranges::contiguous_range`
- ▶ `ranges::borrowed_range`
- ▶ `ranges::view`
- ▶ `ranges::input_range`

Summary

- ▶ `ranges::input_range`, `ranges::output_range`, `ranges::forward_range`,
`ranges::bidirectional_range`, `ranges::random_access_range`,
`ranges::contiguous_range`
- ▶ `ranges::borrowed_range`
- ▶ `ranges::view`
- ▶ `ranges::input_range`
- ▶ `ranges::sized_range`

Summary

- ▶ `ranges::input_range`, `ranges::output_range`, `ranges::forward_range`,
`ranges::bidirectional_range`, `ranges::random_access_range`,
`ranges::contiguous_range`
- ▶ `ranges::borrowed_range`
- ▶ `ranges::view`
- ▶ `ranges::input_range`
- ▶ `ranges::sized_range`
- ▶ `ranges::common_range`

Summary

- ▶ `ranges::input_range`, `ranges::output_range`, `ranges::forward_range`,
`ranges::bidirectional_range`, `ranges::random_access_range`,
`ranges::contiguous_range`
- ▶ `ranges::borrowed_range`
- ▶ `ranges::view`
- ▶ `ranges::input_range`
- ▶ `ranges::sized_range`
- ▶ `ranges::common_range`

When taking a range as an argument, always use universal reference.

Thank you!

Questions?

Books:

- ▶ A Complete Guide to Standard C++ Algorithms
leanpub.com/cpp-algorithms-guide
- ▶ Surviving the C++ Coding Interview
leanpub.com/cpp-coding-interview

Follow Daily bit(e) of C++:

- ▶ linkedin.com/in/simontoth
- ▶ medium.com/@simontoth
- ▶ simontoth.substack.com