Stop war
Stop Putin

C++20
My Favorite Code Examples

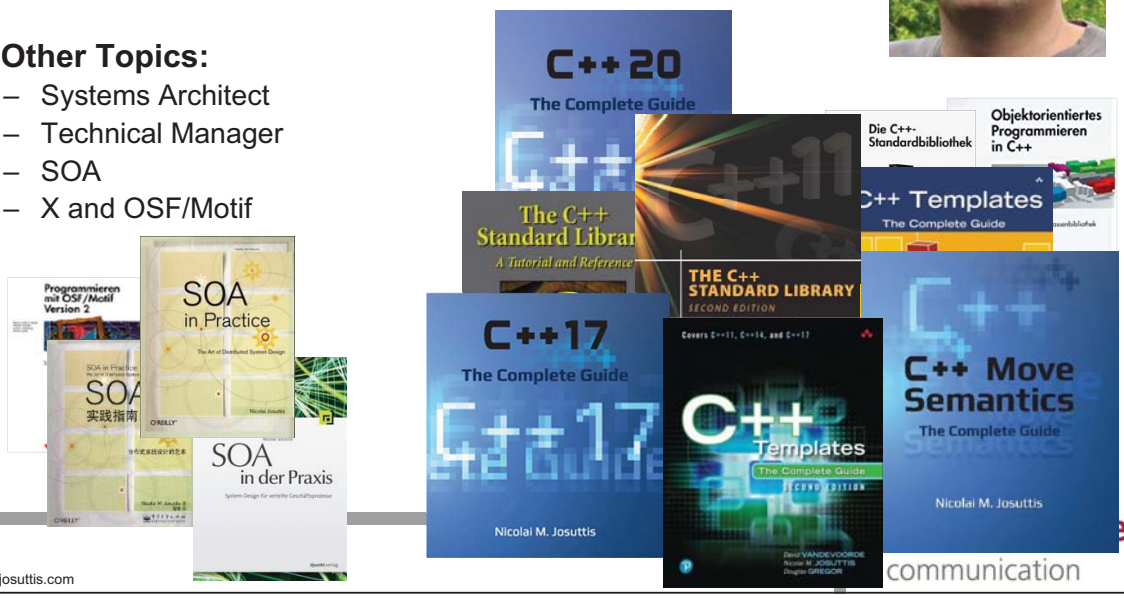
Nicolai M. Josuttis
josuttis.com
@NicoJosuttis

4/22

Stop war
Stop Putin

Nicolai M. Josuttis

- **Independent consultant**
  - Continuously learning since 1962
- **C++:**
  - since 1990
  - ISO Standard Committee since 1997
- **Other Topics:**
  - Systems Architect
  - Technical Manager
  - SOA
  - X and OSF/Motif



C++
©2022 by josuttis.com

Stop war
Stop Putin

C++20

Concepts and Ranges

Stop war
Stop Putin

Let's Add Values Into A Collection

```
template<typename Coll, typename T>
void add(Coll& coll, const T& val)
{
  coll.push_back(val);
}
```

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

add(coll, 42);     // OK
```

Stop war
Stop Putin

auto as Function Parameters

```
void add(auto& coll, const auto& val)
{
  coll.push_back(val);
}
```

"*Abbreviated function template*"

- Generic code
- Equivalent to:

```
template<typename T1, typename T2>
void add(T1& coll, const T2& val) {
  coll.push_back(val);
}
```

- Definition usually in header files
- No `inline` necessary

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

add(coll, 42);     // OK
```


Stop war
Stop Putin

auto as Function Parameters

```
void add(auto& coll, const auto& val)
{
  coll.push_back(val);
}

void add(auto& coll, const auto& val)
{
  coll.insert(val);
}

std::vector<int> coll1;
std::set<int> coll2;

add(coll1, 42);    // ERROR: ambiguous
add(coll2, 42);    // ERROR: ambiguous
```

Stop war
Stop Putin

Concepts as Type Constraints

```
template <typename Coll>
concept HasPushBack = requires (Coll c, Coll::value_type v) {
                        c.push_back(v);
                      };
```

No need for **typename** for a for type of template parameters in **requires** expressions

```
void add(HasPushBack auto& coll, const auto& val)
{
  coll.push_back(val);
}

void add(auto& coll, const auto& val)
{
  coll.insert(val);
}

std::vector<int> coll1;
std::set<int> coll2;

add(coll1, 42);    // OK, uses 1st add() calling push_back()
add(coll2, 42);    // OK, uses 2nd add() calling insert()
```

Equivalent to:

```
template<HasPushBack T1, typename T2>
void add(T1& coll, const T2& val) {
  coll.push_back(val);
}
```

Overload resolution prefers more specialized

Stop war
Stop Putin

Concepts in `requires` Clauses

```
template <typename Coll>
concept HasPushBack = requires (Coll c, Coll::value_type v) {
                        c.push_back(v);
                      };

void add(auto& coll, const auto& val)
requires HasPushBack<...>
{
  coll.push_back(val);
}

void add(auto& coll, const auto& val)
{
  coll.insert(val);
}

std::vector<int> coll1;
std::set<int> coll2;

add(coll1, 42);    // OK, uses 1st add() calling push_back()
add(coll2, 42);    // OK, uses 2nd add() calling insert()
```

Stop war
Stop Putin

Concepts in `requires` Clauses

```
template <typename Coll>
concept HasPushBack = requires (Coll c, Coll::value_type v) {
                        c.push_back(v);
                      };

void add(auto& coll, const auto& val)
requires HasPushBack<decltype(coll)>
{
  coll.push_back(val);
}

void add(auto& coll, const auto& val)
{
  coll.insert(val);
}

std::vector<int> coll1;
std::set<int> coll2;

add(coll1, 42);    // ERROR: can't call insert()
add(coll2, 42);    // OK, uses 2nd add() calling insert()
```

`std::vector<int>&::value_type` is not valid

Stop war
Stop Putin

Concepts in `requires` Clauses

```
template <typename Coll>
concept HasPushBack = requires (Coll c, Coll::value_type v) {
                        c.push_back(v);
                      };

void add(auto& coll, const auto& val)
requires HasPushBack<std::remove_cvref_t<decltype(coll)>>
{
  coll.push_back(val);
}

void add(auto& coll, const auto& val)
{
  coll.insert(val);
}

std::vector<int> coll1;
std::set<int> coll2;

add(coll1, 42);    // OK, uses 1st add() calling push_back()
add(coll2, 42);    // OK, uses 2nd add() calling insert()
```

Stop war
Stop Putin

Concepts in `requires` Clauses

Needs: `#include <ranges>`

```
template <typename Coll>
concept HasPushBack = requires (Coll c, std::ranges::range_value_t<Coll> v) {
                        c.push_back(v);
                      };

void add(auto& coll, const auto& val)
requires HasPushBack<coll>
{
  coll.push_back(val);
}

void add(auto& coll, const auto& val)
{
  coll.insert(val);
}

std::vector<int> coll1;
std::set<int> coll2;

add(coll1, 42);    // OK, uses 1st add() calling push_back()
add(coll2, 42);    // OK, uses 2nd add() calling insert()
```

Don't introduce a concept for each member function
(too fine grained)

Stop war
Stop Putin

`requires`

```
void add(auto& coll, const auto& val)
requires requires { coll.push_back(val); }
{
  coll.push_back(val);
}

void add(auto& coll, const auto& val)
{
  coll.insert(val);
}

std::vector<int> coll1;
std::set<int> coll2;

add(coll1, 42);    // OK, calls push_back()
add(coll2, 42);    // OK, calls insert()
```

- ***Requires expression*** defines ***requirements***
- ***Requires clause*** defines a ***constraint***

Stop war
Stop Putin

requires and Compile-Time if

```
void add(auto& coll, const auto& val)
{
  if constexpr (requires { coll.push_back(val); }) {
    coll.push_back(val);
  }
  else {
    coll.insert(val);
  }
}

std::vector<int> coll1;
std::set<int> coll2;

add(coll1, 42);    // OK, calls push_back()
add(coll2, 42);    // OK, calls insert()
```


Stop war
Stop Putin

requires and Compile-Time if

```
void add(auto& coll, const auto& val)
{
  if constexpr (requires { coll.push_back(val); }) {
    coll.push_back(val);
  }
  else {
    coll.insert(val);
  }
}

std::vector<int> coll1;
std::set<std::string> coll2;

add(coll1, 42);    // OK, calls push_back()
add(coll2, 42);    // ERROR
```

Stop war
Stop Putin

Possible Error Message:

```
prog.cpp:16:10: error: no matching member function for call to 'insert'
    coll.insert(val);
    ~~~~~^~~~~~
prog.cpp:30:1: note: in instantiation of function template specialization
'add<std::set<std::basic_string<char>>, int>' requested here
add(coll2, 42);   // OK, uses 2nd add() calling insert()
^
/include/c++/12.0.1/bits/stl_set.h:509:7: note: candidate function not viable: no
known conversion from 'const int' to 'const
std::set<std::basic_string<char>>::value_type' (aka 'const std::basic_string<char>')
for 1st argument
      insert(const value_type& __x)
      ^
/include/c++/12.0.1/bits/stl_set.h:518:7: note: candidate function not viable: no
known conversion from 'const int' to 'std::set<std::basic_string<char>>::value_type'
(aka 'std::basic_string<char>') for 1st argument
      insert(value_type&& __x)
      ^
…
/include/c++/12.0.1/bits/stl_set.h:603:7: note: candidate function not viable:
requires 2 arguments, but 1 was provided
      insert(const_iterator __hint, node_type&& __nh)
      ^
1 error generated.
```

```
void add(auto& col
{
  if constexpr (re
    coll.push_back
  }
  else {
    coll.insert(va
  }
}

std::vector<int> c
std::set<std::string> coll2;

add(coll1, 42);    // OK, calls push_back()
add(coll2, 42);    // ERROR
```


Stop war
Stop Putin

`requires` and Compile-Time `if`

```
void add(std::ranges::range auto& coll, const auto& val)
{
  if constexpr (requires { coll.push_back(val); }) {
    coll.push_back(val);
  }
  else {
    coll.insert(val);
  }
}

std::vector<int> coll1;
std::set<std::string> coll2;

add(coll1, 42);    // OK, calls push_back()
add(coll2, 42);    // ERROR
```

Stop war
Stop Putin

requires and Compile-Time if

```
template<std::ranges::range Coll, typename T>
void add(Coll& coll, const T& val)
requires std::convertible_to<T, std::ranges::range_value_t<Coll>>
{
  if constexpr (requires { coll.push_back(val); }) {
    coll.push_back(val);
  }
  else {
    coll.insert(val);
  }
}

std::vector<int> coll1;
std::set<std::string> coll2;

add(coll1, 42);    // OK, calls push_back()
add(coll2, 42);    // ERROR
```


Stop war
Stop Putin

requires and Compile-Time if

Possible Error Message (clang):

```
prog.cpp:30:1: error: no matching function for call to 'add'
add(coll2, 42);   // OK, uses 2nd add() calling insert()
^~~
prog.cpp:9:6: note: candidate template ignored: constraints not satisfied [with Coll
= std::set<std::basic_string<char>>, T = int]
void add(Coll& coll, const T& val)
     ^
prog.cpp:10:15: note: because 'std::convertible_to<int,
std::ranges::range_value_t<set<basic_string<char> > > >' evaluated to false
requires std::convertible_to<T, std::ranges::range_value_t<Coll>>
              ^
/include/c++/12.0.1/concepts:72:30: note: because 'is_convertible_v<int,
std::basic_string<char> >' evaluated to false
    concept convertible_to = is_convertible_v<_From, _To>
                             ^
1 error generated.
```

```
template<std::rang
void add(Coll& col
requires std::conv
{
  if constexpr (re
    coll.push_back
  }
  else {
    coll.insert(va
  }
}

std::vector<int> c
std::set<std::string> coll2;

add(coll1, 42);    // OK, calls push_back()
add(coll2, 42);    // ERROR
```

Stop war
Stop Putin

Let's Print Elements

```
template<std::ranges::range Coll, typename T>
void add(Coll& coll, const T& val)
requires std::convertible_to<T, std::ranges::range_value_t<Coll>>
{
  if constexpr (requires { coll.push_back(val); }
    coll.push_back(val);
  }
  else {
    coll.insert(val);
  }
}

std::vector<int> coll1;
std::set<int> coll2;

add(coll1, 42);    // OK, calls push_back()
add(coll2, 42);    // OK, calls insert()
print(coll1);
```


Stop war
Stop Putin

Using Views

```
void print(const std::ranges::input_range auto& coll) {
  for (const auto& elem : coll) {
    std::cout << elem << ' ';
  }
  std::cout << '\n';
}
```

Output:

```
0 8 15 47 11 42
0 8 11 15 42 47
0 8 15
0 8 11
0s 8s 15s
```

```
std::vector<int> coll1{0, 8, 15, 47, 11, 42};
std::set<int> coll2{0, 8, 15, 47, 11, 42};

print(coll1);
print(coll2);
print(coll1 | std::views::take(3));      // print first three elements
print(coll2 | std::views::take(3));      // print first three elements

print(coll1 | std::views::take(3)
            | std::views::transform([](auto v){
                                      return std::to_string(v) + 's';
                                    }));
```

Stop war
Stop Putin

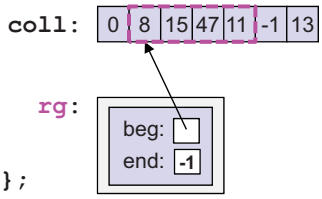
Using Sentinels

```
void print(const std::ranges::input_range auto& coll) {
  for (const auto& elem : coll) {
    std::cout << elem << ' ';
  }
  std::cout << '\n';
}
```

Output:

```
0 8 15 47 11 -1 13
8 15 47 11
8s 15s 47s
```

```
template<auto Val>
struct EndValue {
  bool operator== (auto pos) const {
    return *pos == Val;   // Val is end value
  }
};
```



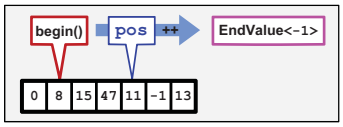

```
std::vector<int> coll1{0, 8, 15, 47, 11, -1, 13};
print(coll1);
std::ranges::subrange rg{coll1.begin() + 1, EndValue<-1>{}};
print(rg);
print(rg | std::views::take(3)
         | std::views::transform([](auto v){
                                   return std::to_string(v) + 's';
                                 }));
```


Stop war
Stop Putin

Using Sentinels

```
void print(const std::ranges::input_range auto& coll) {
  for
    st
  }
  std:
}
```

```
  auto beg = coll1.begin() + 1;
  auto end = EndValue<-1>{};
  for (auto pos = beg; pos != end; ++pos) {
    ...
  }
```



C++20 rewriting converts:
`pos != end`
to:
`!(end == pos)`

```
template<auto Val>
struct EndValue {
  bool operator== (auto pos) const {
    return *pos == Val;   // Val is end value
  }
};
```

```
std::vector<int> coll1{0, 8, 15, 47, 11, -1, 13};
print(coll1);
std::ranges::subrange rg{coll1.begin() + 1, EndValue<-1>{}};
print(rg);
```

Stop war
Stop Putin

Data Structures as Elements

```
void print(const std::ranges::input_range auto& coll) {
  ...
}
```

Literals types with public members
can be Non-Type Template parameters

```
template<auto Val>
struct EndValue {
  bool operator== (auto pos) const {
    return *pos == Val;   // Val is end value
  }
};
```

Output:
```
[3/2/1] [1/2/3]
```

```
struct Coord {
  double x = 0, y = 0, z = 0;
  auto operator<=> (const Coord&) const = default;   // enables all 6 comparison operators
  friend std::ostream& operator<< (std::ostream& strm, const Coord& c) {
    return strm << '[' << c.x << '/' << c.y << '/' << c.z << ']';
  }
};

std::vector points{Coord{3,2,1}, Coord{1,2,3}, Coord{0,0,0}, Coord{4,5,6}};
std::ranges::subrange rg{points.begin(), EndValue<Coord{}>{}};
print(rg);
```


Stop war
Stop Putin

More Complex Elements

```
void print(const std::ranges::input_range auto& coll) {
  ...
}

template<auto Val>
struct EndValue {
  bool operator== (auto pos) const {
    return *pos == Val;   // Val is end value
  }
};
```

Output:
```
[3/2/1] [1/2/3]
[1/2/3] [3/2/1]
```

```
struct Coord {
  double x = 0, y = 0, z = 0;
  auto operator<=> (const Coord&) const = default;   // enables all 6 comparison operators
  friend std::ostream& operator<< (std::ostream& strm, const Coord& c) {
    return strm << '[' << c.x << '/' << c.y << '/' << c.z << ']';
  }
};

std::vector points{Coord{3,2,1}, Coord{1,2,3}, Coord{0,0,0}, Coord{4,5,6}};
std::ranges::subrange rg{points.begin(), EndValue<Coord{}>{}};
print(rg);
std::sort(rg);            // ERROR
std::ranges::sort(rg);    // OK
print(rg);
```

Stop war
Stop Putin

More Complex Elements

```
void print(const std::ranges::input_range auto& coll) {
  …
}
```

```
template<auto Val>
struct EndValue {
  bool operator== (auto pos) const {
    return *pos == Val;   // Val is end value
  }
};
```

Output:

```
[3/2/1] [1/2/3]
[1/2/3] [3/2/1]
```

```
struct Coord {
  double x = 0, y = 0, z = 0;
  auto operator<=> (const Coord&) const = default;   // enables all 6 comparison operators
  friend std::ostream& operator<< (std::ostream& strm, const Coord& c) {
    return strm << std::format("[{}/{}/{}]", c.x, c.y, c.z);
  }
};

std::vector points{Coord{3,2,1}, Coord{1,2,3}, Coord{0,0,0}, Coord{4,5,6}};
std::ranges::subrange rg{points.begin(), EndValue<Coord{}>{}};
print(rg);
std::sort(rg);            // ERROR
std::ranges::sort(rg);    // OK
print(rg);
```


Stop war
Stop Putin

const Propagation

```
void print(const std::ranges::input_range auto& coll) {
  if (*coll.begin() = {}) …;                // OOPS: valid for views => modifies
  if (*coll.cbegin() = {}) …;               // OOPS: no member cbegin() for views
  if (*std::cbegin(coll) = {}) …;           // OOPS: valid for views => modifies
  if (*std::ranges::cbegin(coll) = {}) …;   // OOPS: valid for views => modifies
}

std::vector<int> coll1{0, 8, 15, 47, 11, 42};

print(coll1);                               // elems are const
print(coll2 | std::views::take(3));         // elems are not const
```

- **Const propagation is broken by views**
  - "Works as designed" (reference semantics)
- **`cbegin()` is not available or broken for views**

Stop war
Stop Putin

const Propagation

```
void print23(const std::ranges::input_range auto& coll) {                C++23
  if (*coll.begin() = {}) …;                  // OOPS: valid for views => modifies
  if (*coll.cbegin() = {}) …;                 // member exists but doesn't compile
  if (*std::cbegin(coll) = {}) …;             // OOPS: valid for views => modifies
  if (*std::ranges::cbegin(coll) = {}) …;     // doesn't compile
}

std::vector<int> coll1{0, 8, 15, 47, 11, 42};

print23(coll1);                               // elems are const
print23(coll2 | std::views::take(3));         // elems are not const
```

- **Const propagation is broken by views**
  - "Works as designed" (reference semantics)
- **`cbegin()` is not available or broken for views**
  - Partially fixed with C++23 (`std::cbegin()` is still broken for views)

Stop war
Stop Putin

Let's Print Elements

```
void print(const std::ranges::input_range auto& coll) {
  for (const auto& elem : coll) {
    std::cout << elem << ' ';
  }
  std::cout << '\n';
}

std::vector<int> coll1{0, 8, 15, 47, 11, 42};
std::set<int> coll2{0, 8, 15, 47, 11, 42};

print(coll1);
print(coll2);
print(coll1 | std::views::take(3));       // print first three elements
print(coll2 | std::views::take(3));       // print first three elements

print(coll1 | std::views::drop(3));       // print all but first three elements
print(coll2 | std::views::drop(3));       // ERROR
```

Output:

```
0 8 15 47 11 42
0 8 11 15 42 47
0 8 15
0 8 11
47 11 42
```

error

Stop war
Stop Putin

C++20: Universal/forwarding References for Ranges and Views

- `const` views might not support iterating
  - They might have to modify their state while iterating
  - Use universal/forwarding references when passed by reference

```
template<typename T>
void print(const T& coll) {
  for (const auto& elem : coll) {
    std::cout << elem << '\n';
  }
}
```

No iteration with `const`:
Always: `filter`, `drop_while`, `split`
Sometimes: `drop`, `reverse`, `join`

```
std::vector vec{1, 2, 3, 4, 5};
print(vec);                              // OK
print(vec | std::views::drop(3));        // OK
print(vec | std::views::filter(...));    // ERROR without universal reference
std::list lst{1, 2, 3, 4, 5};
print(lst | std::views::drop(3));        // ERROR without universal reference
```


Stop war
Stop Putin

Let's Print Elements

```
void print(std::ranges::input_range auto&& coll) {
  for (const auto& elem : coll) {
    std::cout << elem << ' ';
  }
  std::cout << '\n';
}
```

Output:

```
0 8 15 47 11 42
0 8 11 15 42 47
0 8 15
0 8 11
47 11 42
15 42 47
```

```
std::vector<int> coll1{0, 8, 15, 47, 11, 42};
std::set<int> coll2{0, 8, 15, 47, 11, 42};

print(coll1);
print(coll2);
print(coll1 | std::views::take(3));      // print first three elements
print(coll2 | std::views::take(3));      // print first three elements

print(coll1 | std::views::drop(3));      // print all but first three elements
print(coll2 | std::views::drop(3));      // OK: print all but first three elements
```

Stop war
Stop Putin

Let's Print Elements

```
void print(std::ranges::input_range auto coll) {
  for (const auto& elem : coll) {
    std::cout << elem << ' ';
  }
  std::cout << '\n';
}
```

Output:

```
0 8 15 47 11 42
0 8 11 15 42 47
0 8 15 47 11 42
0 8 11 15 42 47

47 11 42
15 42 47
```

```
std::vector<int> coll1{0, 8, 15, 47, 11, 42};
std::set<int> coll2{0, 8, 15, 47, 11, 42};

print(coll1);                          // expensive (by value)
print(coll2);                          // expensive (by value)
print(std::views::all(coll1));         // cheap (by reference)
print(coll2 | std::views::all);        // cheap (by reference)

print(coll1 | std::views::drop(3));    // print all but first three elements
print(coll2 | std::views::drop(3));    // OK: print all but first three elements
```


Stop war
Stop Putin

Let's Print Elements

```
void print(std::ranges::view auto coll) {
  for (const auto& elem : coll) {
    std::cout << elem << ' ';
  }
  std::cout << '\n';
}
```

Output:

```
0 8 15 47 11 42
0 8 11 15 42 47

47 11 42
15 42 47
```

```
std::vector<int> coll1{0, 8, 15, 47, 11, 42};
std::set<int> coll2{0, 8, 15, 47, 11, 42};

print(coll1);                          // ERROR
print(coll2);                          // ERROR
print(std::views::all(coll1));         // cheap (by reference)
print(coll2 | std::views::all());      // cheap (by reference)

print(coll1 | std::views::drop(3));    // print all but first three elements
print(coll2 | std::views::drop(3));    // OK: print all but first three elements
```

Stop war
Stop Putin

Example for Pipelines of Range Adaptors

```
int main()
{
  std::map<std::string, int> composer{
    {"Bach", 1685},
    {"Mozart", 1756},
    {"Beethoven", 1770},
    {"Tchaikovsky", 1840},
    {"Chopin", 1810},
    {"Vivaldi", 1678},
  };

  // iterate over the names of the first 3 composers since 1700:
  namespace vws = std::views;
  for (const auto& elem : composer
                          | vws::filter([](const auto& y) {   // since 1700
                                        return y.second >= 1700;
                                      })
                          | vws::take(3)                      // first 3
                          | vws::elements<0>                  // names only
                          ) {
    std::cout << "- " << elem << '\n';
  }
}
```

std::views::elements<0>
does not work for user-defined types

Output:

```
- Beethoven
- Chopin
- Mozart
```


Stop war
Stop Putin

C++20

Compile-Time Computing

Stop war
Stop Putin

Merge a List of Sizes

```
template<typename C, typename... Args>
auto mergeSizes(const C& rg, Args&&... vals)
{
  // initialize vector with passed sizes:
  std::vector<typename C::value_type> v{rg.begin(), rg.end()};

  (... , v.push_back(std::forward<Args>(vals)));   // merge passed sizes

  std::ranges::sort(v);                            // sort sizes

  return v;                                        // return all sizes
}
```

```
std::array a{sizeof(int), sizeof(std::string), sizeof(std::set<int>)};
...
// initialize sorted collection of all sizes:
auto allSizes = mergeSizes(a, sizeof(long), sizeof(Coord));
for(const auto& i : allSizes) {
  std::cout << i << ' ';
}
```

Possible Output:

```
4 32 24 + 8 24
4 8 24 24 32
```


Stop war
Stop Putin

Merge a List of Sizes

```
template<typename C, typename... Args>
auto mergeSizes(const C& rg, Args&&... vals)
{
  // initialize vector with passed sizes:
  std::vector<typename C::value_type> v{rg.begin(), rg.end()};

  (... , v.push_back(std::forward<Args>(vals)));   // merge passed sizes

  std::ranges::sort(v);                            // sort sizes

  return v;                                        // return all sizes
}
```

Error for raw array:
- No `begin()`/`end()`
- No `value_type`

```
std::array a{sizeof(int), sizeof(std::string), sizeof(std::set<int>)};
...
// initialize sorted collection of all sizes:
auto allSizes = mergeSizes(a, sizeof(long), sizeof(Coord));
for(const auto& i : allSizes) {
  std::cout << i << ' ';
}
```

Possible Output:

```
4 32 24 + 8 24
4 8 24 24 32
```

```
int raw[]{0, 8, 15, 132, 4, 77, 3};

auto allSizes2 = mergeSizes(raw, 42, 4);          // ERROR
```

Stop war
Stop Putin

Merge a List of Sizes

```
template<typename C, typename... Args>
auto mergeSizes(const C& rg, Args&&... vals)
{
  // initialize vector with passed sizes:
  std::vector<std::ranges::range_value_t<C>> v{std::ranges::begin(rg),
                                               std::ranges::end(rg)};
  (... , v.push_back(std::forward<Args>(vals)));   // merge passed sizes

  std::ranges::sort(v);                            // sort sizes

  return v;                                        // return all sizes
}
```

```
std::array a{sizeof(int), sizeof(std::string), sizeof(std::set<int>)};
...
// initialize sorted collection of all sizes:
auto allSizes = mergeSizes(a, sizeof(long), sizeof(Coord));
for(const auto& i : allSizes) {
  std::cout << i << ' ';
}
```

Possible Output:

```
4 32 24 + 8 24
4 8 24 24 32
```

```
int raw[]{0, 8, 15, 132, 4, 77, 3};

auto allSizes2 = mergeSizes(raw, 42, 4);          // OK
```


Stop war
Stop Putin

Merge a List of Sizes

```
template<std::ranges::input_range T, std::integral... Args>
auto mergeSizes(const C& rg, Args&&... vals)
{
  // initialize vector with passed sizes:
  std::vector<std::ranges::range_value_t<C>> v{std::ranges::begin(rg),
                                               std::ranges::end(rg)};
  (... , v.push_back(std::forward<Args>(vals)));   // merge passed sizes

  std::ranges::sort(v);                            // sort sizes

  return v;                                        // return all sizes
}
```

```
std::array a{sizeof(int), sizeof(std::string), sizeof(std::set<int>)};
...
// initialize sorted collection of all sizes:
auto allSizes = mergeSizes(a, sizeof(long), sizeof(Coord));
for(const auto& i : allSizes) {
  std::cout << i << ' ';
}
```

Possible Output:

```
4 32 24 + 8 24
4 8 24 24 32
```

Stop war
Stop Putin

Merge a List of Sizes

```
template<std::ranges::input_range T, std::integral... Args>
constexpr auto mergeSizes(const C& rg, Args&&... vals)
{
  // initialize vector with passed sizes:
  std::vector<std::ranges::range_value_t<C>> v{std::ranges::begin(rg),
                                               std::ranges::end(rg)};
  (... , v.push_back(std::forward<Args>(vals)));   // merge passed sizes

  std::ranges::sort(v);                            // sort sizes

  return v;                                        // return all sizes
}
```

```
constexpr std::array a{sizeof(int), sizeof(std::string), sizeof(std::set<int>)};
...
// initialize sorted collection of all sizes:
auto allSizes = mergeSizes(a, sizeof(long), sizeof(Coord));
for(const auto& i : allSizes) {
  std::cout << i << ' ';
}
```

Possible Output:

```
4 32 24 + 8 24
4 8 24 24 32
```

`mergeSizes()` still evaluated at runtime because the result is used at runtime

Stop war
Stop Putin

Merge a List of Sizes at Compile-Time

```
template<std::ranges::input_range T, std::integral... Args>
consteval auto mergeSizes(const C& rg, Args&&... vals)
{
  // initialize vector with passed sizes:
  std::vector<std::ranges::range_value_t<C>> v{std::ranges::begin(rg),
                                               std::ranges::end(rg)};
  (... , v.push_back(std::forward<Args>(vals)));   // merge passed sizes

  std::ranges::sort(v);                            // sort sizes

  return v;                                        // return all sizes
}
```

```
constexpr std::array a{sizeof(int), sizeof(std::string), sizeof(std::set<int>)};
...
// initialize sorted collection of all sizes:
auto allSizes = mergeSizes(a, sizeof(long), sizeof(Coord));   // ERROR
for(const auto& i : allSizes) {
  std::cout << i << ' ';
}
```

Cannot use a compile-time vector at runtime

Stop war
Stop Putin

Merge a List of Sizes at Compile-Time

```
template<std::ranges::input_range T, std::integral... Args>
consteval auto mergeSizes(const C& rg, Args&&... vals)
{
  // initialize vector with passed sizes:
  std::vector<std::ranges::range_value_t<C>> v{std::ranges::begin(rg),
                                               std::ranges::end(rg)};
  (... , v.push_back(std::forward<Args>(vals)));   // merge passed sizes

  std::ranges::sort(v);                            // sort sizes

  // return merges sizes as std::array<>:
  auto sz = std::ranges::size(rg) + sizeof...(vals);
  std::array<std::ranges::range_value_t<C>, sz> ret{};    // ERROR

}
```

"*runtime context*" for `sz` in `std::array<..., sz>`

```
constexpr std::array a{sizeof(int), sizeof(std::string), sizeof(std::set<int>)};
...
// initialize sorted collection of all sizes:
auto allSizes = mergeSizes(a, sizeof(long), sizeof(Coord));
for(const auto& i : allSizes) {
  std::cout << i << ' ';
}
```


Stop war
Stop Putin

Merge a List of Sizes at Compile-Time

```
template<std::ranges::input_range T, std::integral... Args>
consteval auto mergeSizes(const C& rg, Args&&... vals)
{
  // initialize vector with passed sizes:
  std::vector<std::ranges::range_value_t<C>> v{std::ranges::begin(rg),
                                               std::ranges::end(rg)};
  (... , v.push_back(std::forward<Args>(vals)));   // merge passed sizes

  std::ranges::sort(v);                            // sort sizes

  // return merges sizes as std::array<>:
  constexpr auto sz = std::ranges::size(rg) + sizeof...(vals);
  std::array<std::ranges::range_value_t<C>, sz> ret{};    // ERROR

}
```

"*runtime context*" for `size(rg)` because `rg` is passed by reference

```
constexpr std::array a{sizeof(int), sizeof(std::string), sizeof(std::set<int>)};
...
// initialize sorted collection of all sizes:
auto allSizes = mergeSizes(a, sizeof(long), sizeof(Coord));
for(const auto& i : allSizes) {
  std::cout << i << ' ';
}
```

Stop war
Stop Putin

Merge a List of Sizes at Compile-Time

```
template<std::ranges::input_range T, std::integral... Args>
consteval auto mergeSizes(C rg, Args&&... vals)
{
  // initialize vector with passed sizes:
  std::vector<std::ranges::range_value_t<C>> v{std::ranges::begin(rg),
                                               std::ranges::end(rg)};
  (... , v.push_back(std::forward<Args>(vals)));  // merge passed sizes

  std::ranges::sort(v);                           // sort sizes

  // return merges sizes as std::array<>:
  constexpr auto sz = std::ranges::size(rg) + sizeof...(vals);
  std::array<std::ranges::range_value_t<C>, sz> ret{};   // OK
  std::ranges::copy(v, ret.begin());
  return ret;
}
```

```
constexpr std::array a{sizeof(int), sizeof(std::string), sizeof(std::set<int>)};
...
// initialize sorted collection of all sizes:
auto allSizes = mergeSizes(a, sizeof(long), sizeof(Coord));   // OK
for(const auto& i : allSizes) {
  std::cout << i << ' ';
}
```

Possible Output:

```
4 8 24 24 32
```


Stop war
Stop Putin

Merge a List of Sizes at Compile-Time

```
template<std::ranges::input_range T, std::integral... Ar
consteval auto mergeSizes(C rg, Args... vals)
{
  // initialize vector with passed sizes:
  std::vector<std::ranges::range_value_t<C>> v{std::ranges::begin(rg),
                                               std::ranges::end(rg)};
  (... , v.push_back(vals));  // merge passed sizes

  std::ranges::sort(v);       // sort sizes

  // return merges sizes as std::array<>:
  constexpr auto sz = std::ranges::size(rg) + sizeof...(vals);
  std::array<std::ranges::range_value_t<C>, sz> ret{};
  std::ranges::copy(v, ret.begin());
  return ret;
}
```

Pass compile-time parameters by value

```
constexpr std::array a{sizeof(int), sizeof(std::string), sizeof(std::set<int>)};
...
// initialize sorted collection of all sizes:
auto allSizes = mergeSizes(a, sizeof(long), sizeof(Coord));   // OK
for(const auto& i : allSizes) {
  std::cout << i << ' ';
}
```

Possible Output:

```
4 8 24 24 32
```

Stop war
Stop Putin

Merge a List of Sizes at Compile-Time

```
template<std::ranges::input_range T, std::integral... Args>
consteval auto mergeSizes(C rg, Args... vals)
{
  // initialize vector with passed sizes:
  std::vector<std::ranges::range_value_t<C>> v{std::ranges::begin(rg),
                                               std::ranges::end(rg)};
  (... , v.push_back(vals));   // merge passed sizes

  std::ranges::sort(v);        // sort sizes

  // return merges sizes as std::array<>:
  constexpr auto sz = std::ranges::size(rg) + sizeof...(vals);
  std::array<std::ranges::range_value_t<C>, sz> ret{};
  std::ranges::copy(v, ret.begin());
  return ret;
}
```

```
constexpr std::array a{sizeof(int), sizeof(std::string), sizeof(std::set<int>)};
...
// initialize sorted collection of all sizes:
auto allSizes = mergeSizes(a, sizeof(long), sizeof(Coord));   // OK
for(const auto& i : allSizes) {
  std::cout << i << ' ';
}
```

Possible Output:

```
4 8 24 24 32
```


Stop war
Stop Putin

Merge a List of Sizes at Compile-Time

```
template<std::ranges::input_range T, std::integral... Args>
consteval auto mergeSizes(C rg, Args... vals)
{
  // initialize vector with passed sizes:
  std::vector<std::ranges::range_value_t<C>> v{std::ranges::begin(rg),
                                               std::ranges::end(rg)};
  (... , v.push_back(vals));   // merge passed sizes

  std::ranges::sort(v);        // sort sizes

  // return merges sizes as std::array<> plus the computed number of elements:
  constexpr auto maxSz = std::ranges::size(rg) + sizeof...(vals);
  std::array<std::ranges::range_value_t<C>, maxSz> ret{};
  auto res = std::ranges::unique_copy(v, ret.begin());   // remove duplicates
  return std::pair{ret, res.out - ret.begin()};          // return array and size
}
```

Error if no {}

```
constexpr std::array a{sizeof(int), sizeof(std::string), sizeof(std::set<int>)};
...
// initialize sorted collection of all sizes:
auto [arr,sz] = mergeSizes(a, sizeof(long), sizeof(Coord));
auto allSizes = std::views::counted(arr.begin(), sz);
for(const auto& i : allSizes) {
  std::cout << i << ' ';
}
```

Possible Output:

```
4 8 24 32
```

Stop war
Stop Putin

C++20

Calendars and Time Zones

Stop war
Stop Putin

Let's Have a Party

```
#include <iostream>
#include <chrono>
using namespace std::literals;



auto firstDay = 2021y / 1 / 31;
auto lastDay = 12 / 31d / 2021;

for (auto d = firstDay; d <= lastDay; d += std::chrono::months{1}) {
  std::cout << d << ": Party\n";
}
```

Output:

```
2021-01-31: Party
2021-02-31 is not a valid date: Party
2021-03-31: Party
2021-04-31 is not a valid date: Party
2021-05-31: Party
2021-06-31 is not a valid date: Party
2021-07-31: Party
2021-08-31: Party
2021-09-31 is not a valid date: Party
2021-10-31: Party
2021-11-31 is not a valid date: Party
2021-12-31: Party
```

Stop war
Stop Putin

Let's Have a Party

```
#include <iostream>
#include <chrono>
using namespace std::literals;



auto firstDay = 2021y / 1 / 31;
auto lastDay = 12 / 31d / 2021;

for (auto d = firstDay; d <= lastDay; d += std::chrono::months{1}) {
  if (d.ok()) {
    std::cout << d << ": Party\n";
  }
  else {
     std::cout << d.year() / d.month() / 1 + std::chrono::months{1}
               << ": Party\n";
  }
}
```

Output:

```
2021-01-31: Party
2021-03-01: Party
2021-03-31: Party
2021-05-01: Party
2021-05-31: Party
2021-07-01: Party
2021-07-31: Party
2021-08-31: Party
2021-10-01: Party
2021-10-31: Party
2021-12-01: Party
2021-12-31: Party
```


Stop war
Stop Putin

Let's Have a Party

```
#include <iostream>
#include <chrono>
using namespace std::literals;



auto firstDay = 2021y / 1 / std::chrono::last;
auto lastDay = 12 / 31d / 2021;

for (auto d = firstDay; d <= lastDay; d += std::chrono::months{1}) {
  std::cout << d << ": Party\n";
}
```

Output:

```
2021/Jan/last: Party
2021/Feb/last: Party
2021/Mar/last: Party
2021/Apr/last: Party
2021/May/last: Party
2021/Jun/last: Party
2021/Jul/last: Party
2021/Aug/last: Party
2021/Sep/last: Party
2021/Oct/last: Party
2021/Nov/last: Party
2021/Dec/last: Party
```

Stop war
Stop Putin

Let's Have a Party

```
#include <iostream>
#include <chrono>
using namespace std::literals;



auto firstDay = 2021y / 1 / std::chrono::last;
auto lastDay = 12 / 31d / 2021;

for (auto d = firstDay; d <= lastDay; d += std::chrono::months{1}) {
  std::cout << std::format("{:13%B %d}: Party\n", d);
}
```

Output:

```
January 31    : Party
February 28   : Party
March 31      : Party
April 30      : Party
May 31        : Party
June 30       : Party
July 31       : Party
August 31     : Party
September 30  : Party
October 31    : Party
November 30   : Party
December 31   : Party
```


Stop war
Stop Putin

Let's Have a Party

```
#include <iostream>
#include <chrono>
using namespace std::literals;



auto firstDay = 2021y / 1 / std::chrono::last;
auto lastDay = 12 / 31d / 2021;

for (auto d = firstDay; d <= lastDay; d += std::chrono::months{1}) {
  // schedule time (with different time zones):
  auto tp{std::chrono::local_days{d} + 18h + 30min};  // 18:30
  std::chrono::zoned_time timeLocal{std::chrono::current_zone(), tp};
  std::cout << std::format("{0:13%B %d}: Party {0:%X %Z}\n", timeLocal);

  std::chrono::zoned_time timeLA{"America/Los_Angeles", timeLocal};
  std::cout << std::format("{:>19}: {:%X %Z}\n", "LA", timeLA);
}
```

Output:

```
January 31    : Party 18:30:00 MEZ
                   LA: 09:30:00 GMT-8
February 28   : Party 18:30:00 MEZ
                   LA: 09:30:00 GMT-8
March 31      : Party 18:30:00 MESZ
                   LA: 09:30:00 GMT-7
…
September 30  : Party 18:30:00 MESZ
                   LA: 09:30:00 GMT-7
October 31    : Party 18:30:00 MEZ
                   LA: 10:30:00 GMT-7
…
```

Stop war
Stop Putin

C++20

Threads and Concurrency

Stop war
Stop Putin

Concurrent Updates

```
std::map<std::chrono::sys_days, Coord> locs;
auto firstDay = 2021y / 1 / std::chrono::last;
auto lastDay = 12 / 31d / 2021;
for (auto d = firstDay; d <= lastDay; d += std::chrono::months{1}) {
  locs.emplace(d, Coord{});
}

std::vector<std::thread> tasks;
for (int i = 0; i < 5; ++i) {
  tasks.push_back(std::thread{[&locs] {
    while (true) {
      for (auto& [k, c] : locs) {
        c.x += 1, c.y += 1, c.z += 1;
      }
    }
  }});
}
...
```

Several Issues:
- Core dump without `join()`
- Therefore needs exceptions handling
- How to stop the threads?
- Concurrent writes

Stop war
Stop Putin

Concurrent Updates

```
std::map<std::chrono::sys_days, Coord> locs;
auto firstDay = 2021y / 1 / std::chrono::last;
auto lastDay = 12 / 31d / 2021;
for (auto d = firstDay; d <= lastDay; d += std::chrono::months{1}) {
  locs.emplace(d, Coord{});
}

std::vector<std::jthread> tasks;
for (int i = 0; i < 5; ++i) {
  tasks.push_back(std::jthread{[&locs] {
                                 while (true) {
                                   for (auto& [k, c] : locs) {
                                     c.x += 1, c.y += 1, c.z += 1;
                                   }
                                 }
                               }});
}
...

// auto join for all threads when leaving scope
```


Stop war
Stop Putin

Concurrent Updates

```
std::map<std::chrono::sys_days, Coord> locs;
auto firstDay = 2021y / 1 / std::chrono::last;
auto lastDay = 12 / 31d / 2021;
for (auto d = firstDay; d <= lastDay; d += std::chrono::months{1}) {
  locs.emplace(d, Coord{});
}

{std::vector<std::jthread> tasks;
 for (int i = 0; i < 5; ++i) {
   tasks.push_back(std::jthread{[&locs] {
                                  while (true) {
                                    for (auto& [k, c] : locs) {
                                      c.x += 1, c.y += 1, c.z += 1;
                                    }
                                  }
                                }});
 }
 ...
}// auto join for all threads
```

Stop war
Stop Putin

Concurrent Updates

```
std::map<std::chrono::sys_days, Coord> locs;
auto firstDay = 2021y / 1 / std::chrono::last;
auto lastDay = 12 / 31d / 2021;
for (auto d = firstDay; d <= lastDay; d += std::chrono::months{1}) {
  locs.emplace(d, Coord{});
}

{std::vector<std::jthread> tasks;
 for (int i = 0; i < 5; ++i) {
   tasks.push_back(std::jthread{[&locs] (std::stop_token st) {
                                  while (!st.stop_requested()) {
                                    for (auto& [k, c] : locs) {
                                      c.x += 1, c.y += 1, c.z += 1;
                                    }
                                  }
                                }});
 }
 ...
 for (auto& t : tasks) {
   t.request_stop();
 }
}// auto stop request and join for all threads
```

Instead of signal-stop-and-wait for each tread, this signals stop to all and then waits for all

Stop war
Stop Putin

Concurrent Updates

```
std::map<std::chrono::sys_days, Coord> locs;
auto firstDay = 2021y / 1 / std::chrono::last;
auto lastDay = 12 / 31d / 2021;
for (auto d = firstDay; d <= lastDay; d += std::chrono::months{1}) {
  locs.emplace(d, Coord{});
}

{std::vector<std::jthread> tasks;
 for (int i = 0; i < 5; ++i) {
   tasks.push_back(std::jthread{[&locs] (std::stop_token st) {
                                  while (!st.stop_requested()) {
                                    for (auto& [k, c] : locs) {
                                      c.x += 1, c.y += 1, c.z += 1;
                                    }
                                  }
                                }});
 }
 ...
}// auto stop request and join for all threads
print(locs | std::views::values);
```

Possible Output:

```
[597/597/598] [597/597/597] [598/598/598]
[598/598/600] [598/598/601] [602/602/604]
[602/602/604] [605/605/605] [604/604/604]
[605/605/605] [604/604/604] [601/601/604]
```

Stop war
Stop Putin

Concurrent Updates

```
std::map<std::chrono::sys_days, Coord> locs;
auto firstDay = 2021y / 1 / std::chrono::last;
auto lastDay = 12 / 31d / 2021;
for (auto d = firstDay; d <= lastDay; d += std::chrono::months{1}) {
  locs.emplace(d, Coord{});
}

{std::vector<std::jthread> tasks;
 for (int i = 0; i < 5; ++i) {
   tasks.push_back(std::jthread{[&locs] (std::stop_token st) {
                                  while (!st.stop_requested()) {
                                    for (auto& [k, c] : locs) {
                                      std::atomic_ref<Coord> ac{c};
                                      Coord tmp = ac;   // or ac.load();
                                      tmp.x += 1, tmp.y += 1, tmp.z += 1;
                                      ac = tmp;         // or ac.store(tmp);
                                    }
                                  }
                                }});
 }
 ...
}// auto stop request and join for all threads
print(locs | std::views::values);
```

Possible Output:

```
[598/598/598] [600/600/600] [600/600/600]
[600/600/600] [600/600/600] [600/600/600]
[599/599/599] [605/605/605] [604/604/604]
[605/605/605] [605/605/605] [605/605/605]
```


Stop war
Stop Putin

C++20

Summary

Stop war
Stop Putin

What have we used?

C++20:

- Default comparison operators and <=>
- Concepts and requirements
- Ranges and views (incl. spans)
- Coroutines
- `jthread` and stop tokens
- Latches, barriers, semaphores, atomic<> extensions
- Formatted output
- Calendars and time zones
- Modules
- `double`, structs, and lambdas as NTTP's
- Compile-time `new` / strings / vectors
- `consteval` and `constinit`
- UTF-8 characters

Stop war
Stop Putin

Watch Out For More

Nicolai M. Josuttis

www.josuttis.com
nico@josuttis.com
@NicoJosuttis