



How to rangify your code



Talk Recommendations

Talk Recommendations

cppcon 
the C++ conference



ERIC NIEBLER

Ranges for the Standard Library

www.CppCon.org

Calendar Solution

```
int main() {  
    copy(  
        dates_in_year(2015) // 0. Make a range  
        | by_month() // 1. Group the dates by month.  
        | layout_months() // 2. Format the months  
        // strings.  
        | chunk(3) // 3. Group the months  
        // side-by-side.  
        | transpose_months() // 4. Transpose the  
        // of the size-  
        | view::join // 6. Ungroup the s  
        | join_months(), // 7. Join the strings of the transposed  
        // months.  
        ostream_iterator<>(std::cout, "\n")  
    );  
}
```

Composable

Reusable

Works with
infinite ranges

Can show N months
side-by-side

No loops!!!

Correct by
construction.

Copyright Eric Niebler 2015

Talk Recommendations

ACCU
2021
VIRTUAL EVENT

An Overview of Standard Ranges

Tristan Brindle

Talk Recommendations



C++20 Ranges in Practice

Tristan Brindle

20
20



September 13-18

Talk Recommendations



What a View! Building Your Own (Lazy) Range Adaptor (part 1 of 2)

Presenter: Christopher Di Bella

Talk Recommendations – Sy Brand



Register Allocation Explained With My Cats

343 Aufrufe • vor 3 Wochen

Untertitel



Livecoding C++ Ranges: standardisation,...

254 Aufrufe • vor 3 Monaten

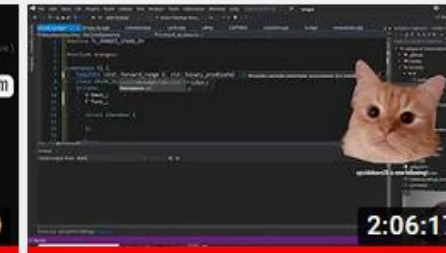
Untertitel



Concurrency vs Parallelism Explained With My Cats

651 Aufrufe • vor 3 Monaten

Untertitel



Livecoding C++ Ranges: chunk_by and chunk_by_key

304 Aufrufe • vor 4 Monaten

Untertitel



Livecoding C++ Ranges: std::ranges::to

346 Aufrufe • vor 4 Monaten

Untertitel



Livecoding C++ Ranges: cycle and cartesian_product

290 Aufrufe • vor 5 Monaten

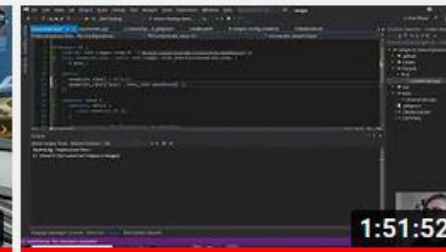
Untertitel



Apex Legends: 30 Seconds Per Frame

256 Aufrufe • vor 5 Monaten

Untertitel



Livecoding C++20 Ranges: enumerate

294 Aufrufe • vor 5 Monaten

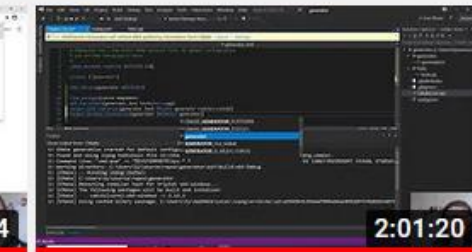
Untertitel



C++ GUIs with coroutines, WinUI3, C++/WinRT and...

1760 Aufrufe • vor 5 Monaten

Untertitel



Implementing Generators with C++20 Coroutines

2002 Aufrufe • vor 6 Monaten

Untertitel



Libraries

Libraries

- [range-v3](#)
- [nanorange](#)
- [TartanLlama/ranges](#)
- [rangesnext](#)
- [boost/range](#)



Rangify?

ranges are algorithms are loops

Rangify?

```
const auto modify = [](const double elem) { return (elem * 2.0) / 10.0; };

const auto vec = std::vector{ 1.0, 2.0, 3.0 };
auto out = std::vector<double>(vec.size());

std::transform(vec.begin(), vec.end(), out.begin(), modify);

std::ranges::transform(vec, out.begin(), modify);

std::ranges::copy(vec | std::views::transform(modify), out.begin());

const auto out2 = vec | ranges::views::transform(modify) | ranges::to<std::vector>();
```



**COMPILER
EXPLORER**



Examples

Max number

```
auto max_number(const std::vector<std::string>& numbers)
{
    auto max = std::numeric_limits<int>::min();
    for (const auto& number : numbers)
    {
        max = std::max(std::stoi(number), max);
    }

    return max;
}
```

```
auto max_number(const std::vector<std::string>& numbers)
{
    return std::ranges::max(numbers
        | std::views::transform([](const auto& number) { return std::stoi(number); }));
}
```



**COMPILER
EXPLORER**

Sliding mean

```
auto sliding_mean(const std::span<const double> rng)
{
    auto out = std::vector<double>(rng.size() - 4);
    for (size_t i = 2; i < rng.size() - 2; ++i)
    {
        out[i - 2] = mean(
            std::array<double, 5>{ rng[i - 2], rng[i - 1], rng[i], rng[i + 1], rng[i + 2] });
    }

    return out;
}

auto sliding_mean(const std::span<const double> rng)
{
    return rng
        | ranges::views::sliding(5)
        | ranges::views::transform(mean)
        | ranges::to<std::vector>();
}
```



COMPILER
EXPLORER

Subtract mean

```
void subtract_mean(
    const std::vector<double>& column_mean,
    boost::multi_array_ref<double, 2> matrix)
{
    for (auto row : matrix)
        for (size_t i = 0; i < column_mean.size(); ++i)
            row[i] -= column_mean[i];
}

void subtract_mean(
    const std::vector<double>& column_mean,
    boost::multi_array_ref<double, 2> matrix)
{
    std::ranges::transform(std::span(matrix.origin(), matrix.num_elements()),
        // matrix | ranges::views::join()
        column_mean | ranges::views::cycle,
        matrix.origin(),
        [](const auto elem, const auto mean) { return elem - mean; });
}
```



COMPILER
EXPLORER

Arange

```
auto next_value(const double step)
{
    return [=](const auto& idx_and_val)
    {
        const auto [idx, val] = idx_and_val;
        return val + static_cast<double>(idx) * step;
    };
}

auto arange(size_t len, double start, double step)
{
    auto out = std::vector<double>(len, start);
    for (size_t i = 0; i < len; ++i)
    {
        out[i] += static_cast<double>(i) * step;
    }
    return out;
}

auto arange(size_t len, double start, double step)
{
    return ranges::views::enumerate(ranges::views::repeat_n(start, len))
        | ranges::views::transform(next_value(step))
        | ranges::to<std::vector>();
}
```



Complex Example

Hondt Method

	party 1 votes: 110		party 2 votes: 85		party 3 votes: 35	
1	(1)	$110 / 1 = 110$	(2)	$85 / 1 = 85$	(6)	$35 / 1 = 35$
2	(3)	$110 / 2 = 55$	(4)	$85 / 2 = 42.5$		$35 / 2 = 17.5$
3	(5)	$110 / 3 = 36.66$	(7)	$85 / 3 = 28.33$		$35 / 3 = 11.66$
4		$110 / 4 = 27.5$		$85 / 4 = 21.25$		$35 / 4 = 8.75$
5		$110 / 5 = 22$		$85 / 5 = 17$		$35 / 5 = 7$
6		$110 / 6 = 18.33$		$85 / 6 = 14.16$		$35 / 6 = 5.83$
7		$110 / 7 = 15.71$		$85 / 7 = 12.14$		$35 / 7 = 5$
	seats: 3		seats: 3		seats: 1	

Hondt Method – pre ranges / C++17

```
auto hondt_method(  
    const std::map<std::string, int>& votes_per_party,  
    const int total_number_of_seats)  
{  
}
```

C++17

Hondt Method – pre ranges / C++17

	party 1 votes: 110	party 2 votes: 85	party 3 votes: 35
1	$110 / 1 = 110$	$85 / 1 = 85$	$35 / 1 = 35$
2	$110 / 2 = 55$	$85 / 2 = 42.5$	$35 / 2 = 17.5$
3	$110 / 3 = 36.66$	$85 / 3 = 28.33$	$35 / 3 = 11.66$
4	$110 / 4 = 27.5$	$85 / 4 = 21.25$	$35 / 4 = 8.75$
5	$110 / 5 = 22$	$85 / 5 = 17$	$35 / 5 = 7$
6	$110 / 6 = 18.33$	$85 / 6 = 14.16$	$35 / 6 = 5.83$
7	$110 / 7 = 15.71$	$85 / 7 = 12.14$	$35 / 7 = 5$

Hondt Method – pre ranges / C++17

```
auto hondt_method(  
    const std::map<std::string, int>& votes_per_party,  
    const int total_number_of_seats)  
{  
    auto proportional_votes = std::vector<std::pair<std::string, double>>();  
    for (int i = 1; i < total_number_of_seats + 1; ++i)  
    {  
        for (const auto& [party, number_of_votes] : votes_per_party)  
        {  
            proportional_votes.push_back({ party, static_cast<double>(number_of_votes) / i });  
        }  
    }  
}
```

C++17

Hondt Method – pre ranges / C++17

	party 1 votes: 110		party 2 votes: 85		party 3 votes: 35	
1	(1)	$110 / 1 = 110$	(2)	$85 / 1 = 85$	(6)	$35 / 1 = 35$
2	(3)	$110 / 2 = 55$	(4)	$85 / 2 = 42.5$		$35 / 2 = 17.5$
3	(5)	$110 / 3 = 36.66$	(7)	$85 / 3 = 28.33$		$35 / 3 = 11.66$
4		$110 / 4 = 27.5$		$85 / 4 = 21.25$		$35 / 4 = 8.75$
5		$110 / 5 = 22$		$85 / 5 = 17$		$35 / 5 = 7$
6		$110 / 6 = 18.33$		$85 / 6 = 14.16$		$35 / 6 = 5.83$
7		$110 / 7 = 15.71$		$85 / 7 = 12.14$		$35 / 7 = 5$

Hondt Method – pre ranges / C++17

C++17

```
auto hondt_method(
    const std::map<std::string, int>& votes_per_party,
    const int total_number_of_seats)
{
    auto proportional_votes = std::vector<std::pair<std::string, double>>();
    for (int i = 1; i < total_number_of_seats + 1; ++i)
    {
        for (const auto& [party, number_of_votes] : votes_per_party)
        {
            proportional_votes.push_back({ party, static_cast<double>(number_of_votes) / i });
        }
    }

    std::sort(proportional_votes.begin(),
              proportional_votes.end(),
              [](const auto& rhs, const auto& lhs)
              {
                  return rhs.second > lhs.second;
              });
    proportional_votes.resize(total_number_of_seats);
}
```


Hondt Method – pre ranges / C++17

	party 1 votes: 110		party 2 votes: 85		party 3 votes: 35	
1	(1)	$110 / 1 = 110$	(2)	$85 / 1 = 85$	(6)	$35 / 1 = 35$
2	(3)	$110 / 2 = 55$	(4)	$85 / 2 = 42.5$		$35 / 2 = 17.5$
3	(5)	$110 / 3 = 36.66$	(7)	$85 / 3 = 28.33$		$35 / 3 = 11.66$
4		$110 / 4 = 27.5$		$85 / 4 = 21.25$		$35 / 4 = 8.75$
5		$110 / 5 = 22$		$85 / 5 = 17$		$35 / 5 = 7$
6		$110 / 6 = 18.33$		$85 / 6 = 14.16$		$35 / 6 = 5.83$
7		$110 / 7 = 15.71$		$85 / 7 = 12.14$		$35 / 7 = 5$
	seats: 3		seats: 3		seats: 1	

Hondt Method – pre ranges / C++17

C++17

```
auto hondt_method(  
    const std::map<std::string, int>& votes_per_party,  
    const int total_number_of_seats)  
{  
    ...  
  
    proportional_votes.resize(number_of_seats);  
  
    auto distribution = std::map<std::string, int>();  
    for (const auto& [party, number_of_votes] : votes_per_party)  
    {  
        const auto count = std::count_if(proportional_votes.begin(),  
                                         proportional_votes.end(),  
                                         [&](const auto& votes)  
                                         {  
                                             return votes.first == party;  
                                         });  
        distribution.insert({ party, count });  
    }  
  
    return distribution;  
}
```

```
auto hondt_method(const std::map<std::string, int>& votes_per_party, const int total_number_of_seats)
```

```
{
```

```
    auto proportional_votes = std::vector<std::pair<std::string, double>>();
```

```
    for (int i = 1; i < total_number_of_seats + 1; ++i)
```

```
    {
```

```
        for (const auto& [party, number_of_votes] : votes_per_party)
```

```
        {
```

```
            proportional_votes.push_back({ party, static_cast<double>(number_of_votes) / i });
```

```
        }
```

```
    }
```

calculate proportional votes

```
    std::sort(proportional_votes.begin(), proportional_votes.end(), [](const auto& rhs, const auto& lhs)
```

```
    {
```

```
        return rhs.second > lhs.second;
```

```
    });
```

```
    proportional_votes.resize(total_number_of_seats);
```

sort and cut

```
    auto distribution = std::map<std::string, int>();
```

```
    for (const auto& [party, number_of_votes] : votes_per_party)
```

```
    {
```

```
        const auto count = std::count_if(proportional_votes.begin(), proportional_votes.end(),
```

```
            [&](const auto& votes)
```

```
            {
```

```
                return votes.first == party;
```

```
            });
```

```
        distribution.insert({ party, count });
```

```
    }
```

count seats per party

```
    return distribution;
```

```
}
```

Hondt Method – rangified

```
auto proportional_votes = std::vector<std::pair<std::string, double>>();
for (int i = 1; i < total_number_of_seats + 1; ++i)
{
    for (const auto& [party, number_of_votes] : votes_per_party)
    {
        proportional_votes.push_back({ party, static_cast<double>(number_of_votes) / i });
    }
}
```

C++17

C++20

```
auto seat_divisors = ranges::views::ints(1, total_number_of_seats + 1);
auto proportional_votes = ranges::views::cartesian_product(votes_per_party, seat_divisors)
```

Hondt Method – rangified

```
{ { "party_1", 110 }, 1 }, { { "party_2", 85 }, 1 }, { { "party_3", 35 }, 1 },  
{ { "party_1", 110 }, 2 }, { { "party_2", 85 }, 2 }, { { "party_3", 35 }, 2 },  
{ { "party_1", 110 }, 3 }, { { "party_2", 85 }, 3 }, { { "party_3", 35 }, 3 },  
...
```

Hondt Method – rangified

```
auto proportional_votes = std::vector<std::pair<std::string, double>>();
for (int i = 1; i < total_number_of_seats + 1; ++i)
{
    for (const auto& [party, number_of_votes] : votes_per_party)
    {
        proportional_votes.push_back({ party, static_cast<double>(number_of_votes) / i });
    }
}
```

C++17

```
const auto divide_votes_by_seat_divisors = [] (const auto& votes_and_divisor)
{
    const auto& [party_and_vote, divisor] = votes_and_divisor;
    return party_and_proportion{ party_and_vote.first,
        party_and_vote.second / static_cast<double>(divisor) };
};

auto seat_divisors = ranges::views::ints(1, total_number_of_seats + 1);
auto proportional_votes = ranges::views::cartesian_product(votes_per_party, seat_divisors)
    | ranges::views::transform(divide_votes_by_seat_divisors)
    | ranges::to<std::vector>();
```

C++20

Hondt Method – rangified

```
std::sort(proportional_votes.begin(),  
          proportional_votes.end(),  
          [](const auto& rhs, const auto& lhs)  
          {  
              return rhs.second > lhs.second;  
          });  
proportional_votes.resize(total_number_of_seats);
```

C++17

```
std::ranges::sort(proportional_votes, [](const auto& rhs, const auto& lhs)  
{  
    return rhs.second > lhs.second;  
});  
proportional_votes.resize(total_number_of_seats);
```

C++20

Hondt Method – rangified

```
std::sort(proportional_votes.begin(),  
          proportional_votes.end(),  
          [](const auto& rhs, const auto& lhs)  
          {  
            return rhs.second > lhs.second;  
          });  
proportional_votes.resize(total_number_of_seats);
```

C++17

```
std::ranges::sort(proportional_votes, [](const auto& rhs, const auto& lhs)  
                 {  
                   return rhs.proportion > lhs.proportion;  
                 });  
proportional_votes.resize(total_number_of_seats);
```

C++20

Hondt Method – rangified

```
std::sort(proportional_votes.begin(),  
          proportional_votes.end(),  
          [](const auto& rhs, const auto& lhs)  
          {  
              return rhs.second > lhs.second;  
          });  
proportional_votes.resize(total_number_of_seats);
```

C++17

```
std::ranges::sort(proportional_votes, std::greater(), &party_and_proportion::proportion);  
proportional_votes.resize(total_number_of_seats);
```

C++20

Hondt Method – rangified

```
std::sort(proportional_votes.begin(),  
          proportional_votes.end(),  
          [](const auto& rhs, const auto& lhs)  
          {  
              return rhs.second > lhs.second;  
          });  
proportional_votes.resize(total_number_of_seats);
```

C++17

```
proportional_votes |= ranges::actions::sort(std::greater(),  
                                           &party_and_proportion::proportion);  
proportional_votes.resize(total_number_of_seats);
```

C++20

Hondt Method – rangified

```
std::sort(proportional_votes.begin(),  
          proportional_votes.end(),  
          [](const auto& rhs, const auto& lhs)  
          {  
              return rhs.second > lhs.second;  
          });  
proportional_votes.resize(total_number_of_seats);
```

C++17

```
proportional_votes |= ranges::actions::sort(std::greater(),  
                                           &party_and_proportion::proportion);
```

C++20

Hondt Method – rangified

```
auto distribution = std::map<std::string, int>();
for (const auto& [party, number_of_votes] : votes_per_party)
{
    const auto count = std::count_if(proportional_votes.begin(), proportional_votes.end(),
                                     [&](const auto& votes)
    {
        return votes.first == party;
    });
    distribution.insert({ party, count });
}
```

C++17

```
return votes_per_party
    | ranges::views::keys
    | ranges::views::transform(count_seats_per_party(proportional_votes, total_number_of_seats))
    | ranges::to<std::map>();
```

C++20

Hondt Method – rangified

C++20

```
auto calculate_number_of_seats(  
    const std::vector<party_and_proportion>& proportional_votes,  
    const int total_number_of_seats,  
    const std::string_view party)  
{  
    return ranges::count_if(proportional_votes | ranges::views::take(total_number_of_seats),  
                           [&](const auto& party_and_votes)  
                           {  
                               return party_and_votes.party == party;  
                           });  
}  
  
auto count_seats_per_party(  
    const std::vector<party_and_proportion>& proportional_votes,  
    const int total_number_of_seats)  
{  
    return [&, number_of_seats](const auto& party)  
    {  
        const auto seats =  
            calculate_number_of_seats(proportional_votes, total_number_of_seats, party);  
        return std::pair{ party, seats };  
    };  
}
```

Hondt Method – rangified

C++20

```
auto hondt_method(  
    const std::map<std::string, int>& votes_per_party,  
    const int total_number_of_seats)  
{  
    auto seat_divisors = ranges::views::ints(1, total_number_of_seats + 1);  
    auto proportional_votes = ranges::views::cartesian_product(votes_per_party, seat_divisors)  
        | ranges::views::transform(divide_votes_by_seat_divisors)  
        | ranges::to<std::vector>();  
  
    proportional_votes |= ranges::actions::sort(std::greater(),  
        &party_and_proportion::proportion);  
  
    return votes_per_party  
        | ranges::views::keys  
        | ranges::views::transform(count_seats_per_party(proportional_votes,  
            total_number_of_seats))  
        | ranges::to<std::map>();  
}
```



COMPILER
EXPLORER



Summary

Questions?

Tina Ulbrich

@_Yulivee_

ROSEN Technology and Research Center GmbH

Talks

- <https://www.youtube.com/c/SyBrandPlusCats/videos>
- <https://youtu.be/YWayW5ePpkY>
- https://youtu.be/d_E-VLyUnzc
- <https://youtu.be/d9qDEEJFwNc>
- <https://youtu.be/mFUXNMfaciE>

Libraries

- <https://en.cppreference.com/w/cpp/ranges>
- <https://github.com/cor3ntin/rangesnext>
- <https://github.com/tcbrindle/NanoRange>
- <https://github.com/TartanLlama/ranges>
- <https://github.com/ericniebler/range-v3>
- https://www.boost.org/doc/libs/1_75_0/libs/range/doc/html/index.htm
|

Resources

- https://en.wikipedia.org/wiki/D%27Hondt_method
- <https://en.cppreference.com/w/cpp/ranges>
- <https://ericniebler.github.io/range-v3/>
- <https://github.com/cor3ntin/rangesnext>