BME TMIT
2022

14/10

Németh Gábor

C++20 range-ek

<ranges>

- C++20
  - ranges-v3

- lusta kiértékelés
  - `std::vector vec{1, 2, 3, 4, 5, 6};`
  - `auto v = std::views::reverse(vec);`

csak igény esetén
értékelődik ki

  - `std::cout << *v.begin() << '\n';`

Motiváció

- `std::sort(v.begin() + 5, v.end());`
- `std::sort(v.rbegin(), v.rend());`
- `std::sort(v.rbegin() + 5, v.rend());`

- `std::ranges::sort(std::views::drop(v, 5));`
- `std::ranges::sort(std::views::reverse(v));`
- `std::ranges::sort(std::views::drop(std::views::reverse(v), 5));`

Nézetek és kombinálásuk

- nézetek
  - általában egy másik nézet segítségével vannak definiálva
  - azon egy műveletet hajtanak végre

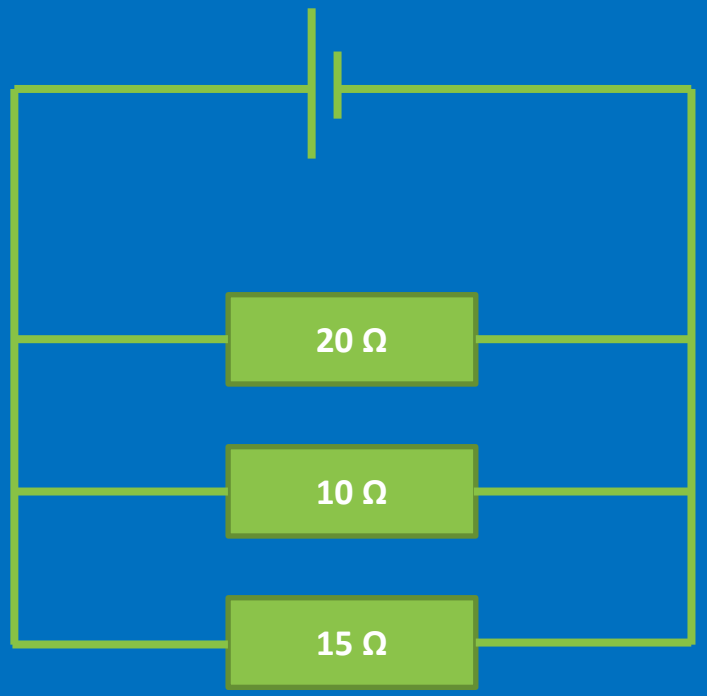
- |
  - `std::vector vec{1, 2, 3, 4, 5, 6};`
  - `auto v = vec | std::views::reverse | std::views::drop(2);`
  - `std::cout << *v.begin() << '\n';`

filter és transform

- std::views::filter
  - range elemeinek szűrés
  - `std::vector{ 1,2,3 } | std::views::filter([](int n){ return n % 2; })`

- std::views::transform
  - range elemeinek transzformálása
  - `std::vector{ 1,2,3 } | std::views::transform([](int n){ return 2 * n; })`

ex_1: Ellenállások




```
#include <ranges>
#include <vector>

…

const auto r = {20,10,15};
const auto r_inv = r | std::views::transform([](int x){ return 1.0 / x; });
const auto val = 1.0 / std::accumulate(std::ranges::cbegin(r_inv), std::ranges::cend(r_inv), 0.0);
```

take és take_while

- std::views::take
  - megadott számú elem
  - `std::ranges::for_each(p | std::views::take(6), [](char ch){ std::cout << ch; });`

- std::views::take_while
  - amíg a megadott predikátum teljesül
  - `years | std::views::take_while([](int y) { return y <= 2020; }))`

std::views::iota

```
for (int i : std::ranges::iota_view{1, 10})
    std::cout << i << ' ';
std::cout << '\n’;
```

```
for (int i : std::views::iota(1, 10))
    std::cout << i << ' ';
std::cout << '\n’;
```

```
for (int i : std::views::iota(1) | std::views::take(9))
    std::cout << i << ' ';
std::cout << '\n';
```

1 2 3 4 5 6 7 8 9

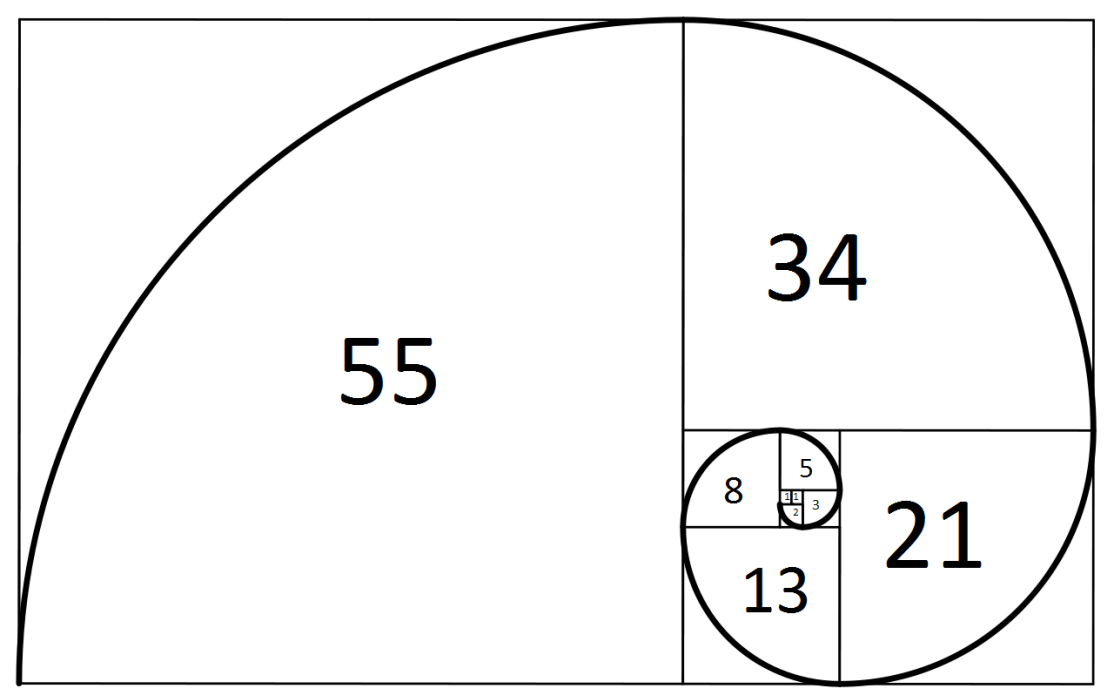
ex_2: Bin2Dec

1	1	1	0

$$1 * 2^3 + 1 * 2^2 + 1 * 2^1 + 0 * 2^0$$

```
auto const v = std::vector<std::uint8_t> {1,1,1,0};
auto r_rev = v | std::views::reverse;
auto r_int = std::views::iota(0lu, v.size());
auto r_pow = r_int | std::views::transform([](int x){ return 1 << x; });
auto val = std::inner_product(std::cbegin(r_rev), std::cend(r_rev),std::cbegin(r_pow),0);
```

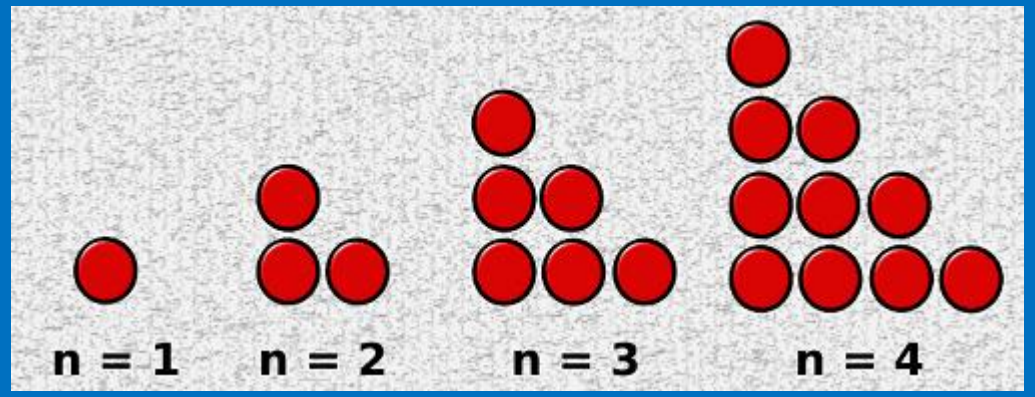
ex_3: std::ranges::generate: Fibonacci




```
std::array<int, 10> vr;

std::ranges::generate(vr, [p = std::pair{0,1}] () mutable {
    auto [a0,b0] = p;
    p = {b0, a0 + b0};
    return a0;
});
```

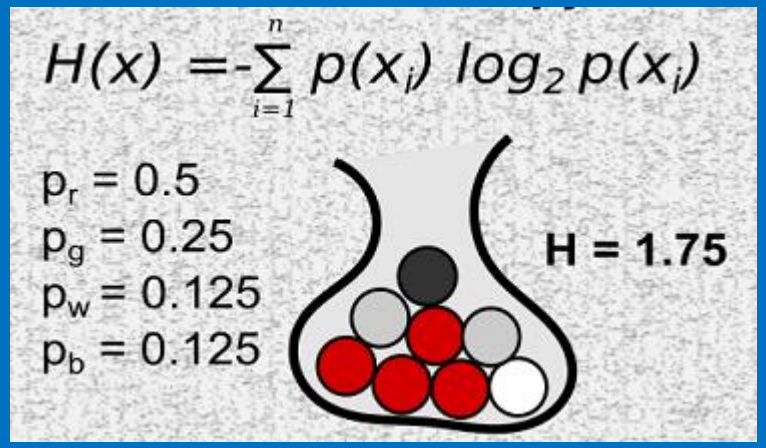
ex_4: Triangular sequence




```
auto res = std::views::iota(1) | std::views::transform([](int n){ return n*(n+1)/2; }) | std::views::take(5);

for(const auto &e : res)
    std::cout << e << ' ';
```

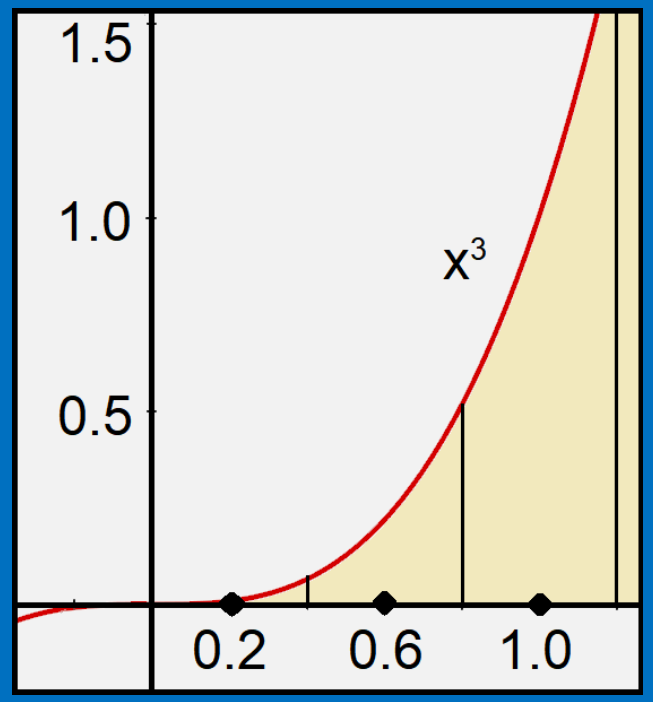
ex_5: Entrópia




```
auto const v = std::vector{0.5,0.25,0.125,0.125};

auto r_p_logp =  v | std::views::transform([](auto p){ return -p * std::log2(p); });
auto val = std::accumulate(std::ranges::cbegin(r_p_logp), std::ranges::cend(r_p_logp), 0.0);
```

ex_6: Integrálás



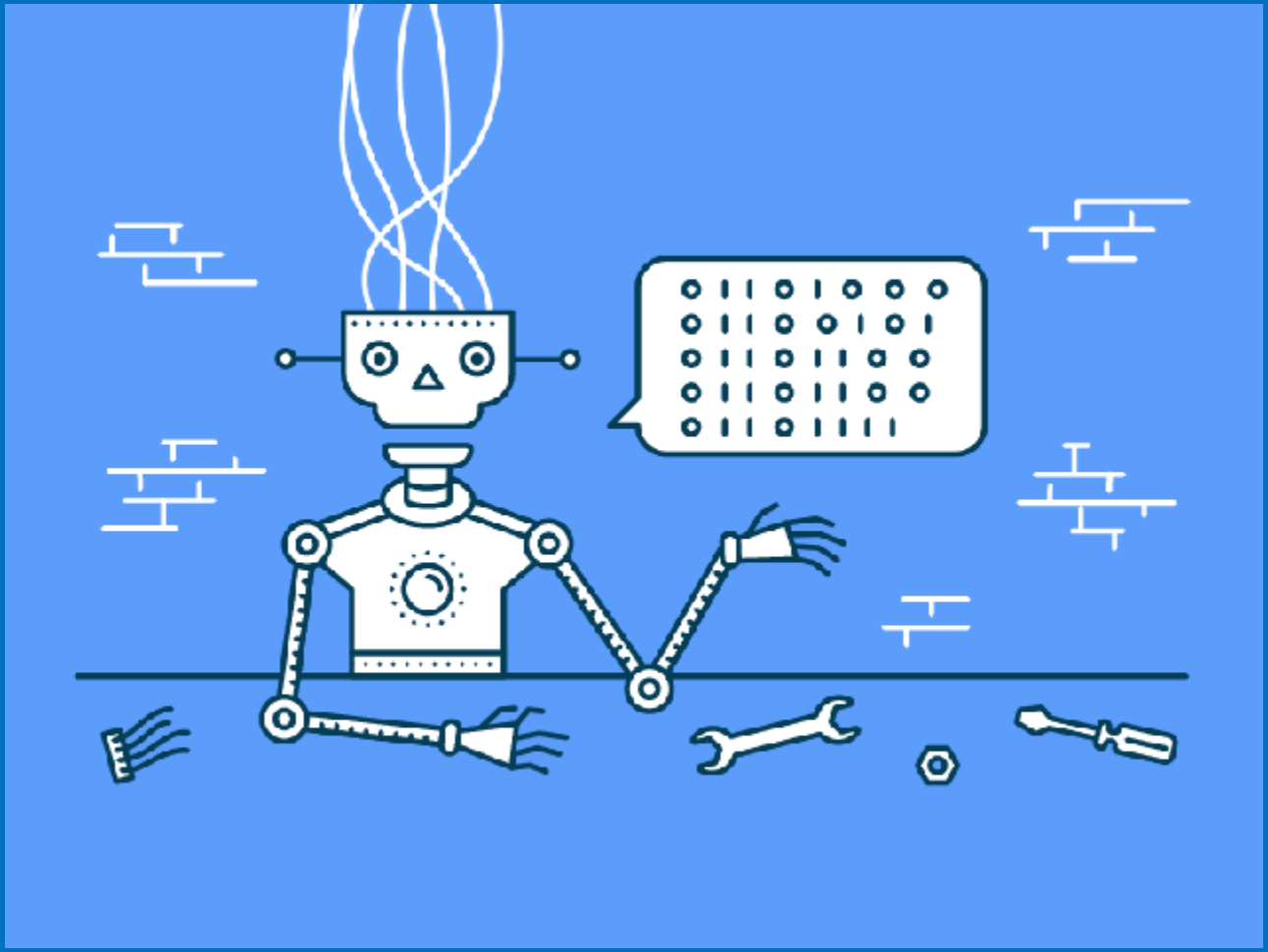

```
auto steps = 500;
auto a = 0.0;
auto b = 1.0;
auto dx = (b - a) / steps;
auto fun = [](double x){ return 4 / (1 + x * x); };

auto r_int = std::views::iota(0, steps);
auto r_pos = r_int | std::views::transform([dx](int i){ return dx * (0.5 + i); });

auto curve = r_pos | std::views::transform(fun);

auto area = dx*std::accumulate(std::ranges::cbegin(curve), std::ranges::cend(curve), 0.0);
```

ex_7: fmap



ex_7: fmap

std::make_pair két std::view::iota-ból

körrizés

```
[](int i) { return [i](int j) { return std::make_pair(i, j); };}
```

alkalmazás az
első range-re

```
auto app = std::view::iota(10, 11) | std::views::transform([](int i) {
    return [i](int j) { return std::make_pair(i, j); }
});
```

Applikatív funktor

```
auto app = std::views::iota(10, 12) | std::views::transform([](int i) {
    return [i](int j) { return std::make_pair(i, j); };
});
```

…

```
app = applikatív funktor
/*   lásd 8. előadás   */
```

```
for (const auto& f : views::fmap_product(app, std::views::iota(10, 12))) {
    std::cout << f.first << ", " << f.second << '\n';
}

auto v = std::vector<int>{ 5,6,7 };
for (const auto& f : views::fmap_product(app, v)) {
    std::cout << f.first << ", " << f.second << '\n';
}
```

!

ex_7

Köszönöm a figyelmet!

Folytatjuk…