Biblioteka <ranges>

czyli coś o zakresach…

Wojciech Palka
Natalia Mendera

#include <ranges>

Zakresy pojawiły się w standardzie C++20. W naszych kompilatorach musimy w ustawieniach wybrać najnowszą wersję standardu języka, aby wszystko działało poprawnie. Bibliotekę tą zaimplementował Eric Niebler. Korzystanie z niej ma swoje zastosowanie w połączeniu z biblioteką <algorithm>, która zawiera wiele algorytmów uogólnionych. Aby posortować dane kontenera przy użyciu funkcji sort(...), musimy zawsze przekazywać iterator na początek i koniec danych do posortowania. Zazwyczaj jednak chcemy posortować cały kontener, przekazywanie zatem parametrów kontener.begin() oraz kontener.end() wydaje się zbędne. Dzięki bibliotece <ranges> możemy tego uniknąć, dzięki czemu nasz kod staje się bardziej przejrzysty.

```cpp
#include <vector>
#include <algorithm>
#include <ranges>
#include <iostream>

int main() {
    std::vector<int> vec = { 1, 5, 3, 0, -2, 5 };
    std::sort(vec.begin(), vec.end());
    std::for_each(vec.begin(), vec.end(), [](const auto i) {
        std::cout << i << ' ';
        });
    //Z użyciem <ranges>
    std::ranges::sort(vec);
    std::ranges::for_each(vec, [](const auto i){
        std::cout << i << ' ';
    });
}
```

Views

Widoki pozwalają nam wygodny i czytelny sposób otrzymać z kolekcji elementy, które chcemy, oraz przekształcić je jak chcemy, a wszystko element po elemencie. Przy ich użyciu stosuje się nową składnię, mianowicie aby zapisać wykonywane na kontenerze operacje oddzielamy funkcje symbolem | (pionowej kreski). Zaletą korzystania z widoków jest oszczędność czasu na przekształcaniu i wyłuskiwaniu potrzebnych nam elementów kontenera, ponieważ wszystkie operacje wykonują się na raz element po elemencie, nie musimy wielokrotnie przechodzić przez kontener. Kolejność wykonywania funkcji na danym elemencie zależy od ich zapisu- wykonują się od lewej do prawej.

```cpp
#include<ranges>
#include<vector>
#include<algorithm>
#include<iostream>

int main()
{
    std::vector<int> input = { 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10 };
    std::vector<int> intermediate, output1;

    std::copy_if(input.begin(), input.end(), std::back_inserter(intermediate), [](const int i) { return i % 3 == 0; });
    std::transform(intermediate.begin(), intermediate.end(), std::back_inserter(output1), [](const int i) {return i * i; });

    //przy użyciu <ranges>
    std::vector<int> vec = { 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10 };

    auto output2 = vec | std::views::filter([](const int n) {return n % 3 == 0; }) | std::views::transform([](const int n) {return n * n; });
    for (int i : output2)
    {
        std::cout << i << std::endl;
    }
}
```

Dostępne operacje na widokach

r | view::all - tworzy zakres z elementów, które mogą już być widokiem, chociaż r nie może być prawą-referencją

r | view::filter(pred)- zwraca zakres, dla którego wszystkie elementy spełniają warunek predykatu

r | view::transform(fn)- zwraca zakres, dla którego wszystkie elementy z r, zostały już przetransformowane

r | view::reverse- zwraca zakres elementów r w odwrotnej kolejności

r | view::take(n)- zwraca zakres zawierający pierwsze n elementów z r, a jeśli r ma mniej -wtedy wszystkie elementy

r | view::join- łączy zakresy w jeden zakres

r | view::drop(n)- zwraca zakres bez pierwszych n elementów

r | view::drop_while- zwraca zakres, w którym pierwsze elementy są pomijane do czasu natrafienia na element spełniający warunek predykatu

views::counted(r.begin(),count)- zwraca zakres zawierający count elementów od wskazanego iteratora

```cpp
#include <ranges>
#include <iostream>
#include <vector>

int main()
{
    std::vector<int> vec = { 1,2,3,4,5 };
    for (int i : vec
        | std::views::drop(2)
        | std::views::drop_while([](int n) {return n < 4; }))
    {
        std::cout << i << " ";
    }
}
```

Generowanie widoków

views::empty- zwraca pusty zakres

views::single(x)- zwraca zakres z jednym elementem

views::iota(range_from,range_to)- zwraca zakres wygenerowany z elementów podanego przedziału zwiększanych o jeden

```cpp
#include <ranges>
#include <iostream>
#include <vector>

int main()
{
    for (int i : std::views::iota(1, 10))
        std::cout << i << ' ';
}
```

Źródła:

https://docs.microsoft.com/

https://cpp0x.pl/

kanał Fureeish na plaformie YouTube

Dziękujemy za uwagę :)