C++: Koncepty

Piotr Bartman-Szwarc

Wprowadzenie

Koncept

predykat czasu kompilacji sprawdzający wymagania (constraints) odnośnie argumentów szablonu dostępny w C++20

Wspierane przez:

- gcc 10
- clang 10
- MSVC 19.30

DEMO

https://godbolt.org/z/4d85M7

Jak definujemy wymagania?

```
1  template<typename T>
2  T f(T x) requires Concept<T> {return x;}
3
4  template<typename T> requires SomeConcept<T>
5  T f(T x) {return x;}
6
7  template<typename T> requires SomeConcept<T>
8                               && AnotherConcept<T>
9  T f(T x) {return x;}
10
11 template<typename T> requires SomeConcept<T>
12                              || AnotherConcept<T>
13 T f(T x) {return x;}
```

NTTP (Non-Type Template Parameter) z wymaganiami

```cpp
#include <array>

template <typename T, std::size_t N>
    requires (N > 2)
auto third_element(std::array<T, N> my_array) {
    return my_array[2];
}

int main() {
    auto my_array = std::array<int, 3>({1,2,3});
    return third_element(my_array);
}
```

Koncepty standardowe

https://en.cppreference.com/w/cpp/named_req

DEMO

https://godbolt.org/z/Cov-tp

Jak definiujemy koncept

```
1 template <typename T>
2 concept ConceptName = OtherC<T> || trait<T>::value;
```

```
1 template <typename T>
2 concept Recursion = Recursion<T>;  // :(
3
4 template <typename T> requires SomeConcept<T>  // :(
5 concept MyConcept = OtherConcept<T>;
```

Jak definiujemy koncept poprzez wymagania

```
template <typename T>
concept Addable = requires (T a, T b) {
    a + b;
};
```

```
template <typename T>
concept HasNestedType = requires {
    typename T::value_type;
    typename T::size_type;
};
```

```
template<typename T> concept AddableLikeFloats =
requires (T a, T b) {
 {a + b} noexcept -> std::convertible_to<float>;
};
```

Przykłady

```
template <typename T>
concept superfluous = true;
```

```
template <typename X, typename Y>
concept they_are_mathsy = requires(X x, Y y)
{
    { x * y };
    { x / y };
    { x + y };
    { x - y };
};
```

NTTP (Non-Type Template Parameter) z konceptami

```
1  template <int N, int M>
2  concept DiffOfInts = (N - M == 0);
3
4  template <int N, int M>
5  int sum() requires DiffOfInts<N,M> {
6      return N+M;
7  }
8
9  int main() {
10     std::cout << sum<7,7>();
11     // std::cout << sum<7,8>();
12     return 0;
13 }
```

Zadanie 1

https://godbolt.org/z/vTqmxH

Zadanie 2

https://godbolt.org/z/ro7deeW1T

auto z konceptami

```
1 std::floating_point auto x = 5.0;
2 std::floating_point auto divide(
3     std::floating_point auto first,
4     std::floating_point auto second)
5 {
6     return first / second;
7 }
8 std::floating_point auto my_result = divide(x,y)
```

auto z konceptami

```
1 template <typename G, knight_concept K>
2 void murder_jim(G&& game_mode, K&& knight)
3 {
4     dragon_concept auto jim = find_jim(game_mode);
5 
6     knight.murder_dragon(jim);
7 }
```

Dlaczego używać konceptów?

- bez konceptów wymagania są ukryte:
  - w ciele funkcji
  - w dokumentacji
  - w "boilerplate code" z pomocą `enable_if` i `void_t`
- zwiększa to czytelność kodu (przejrzyste interfejsy) zerowym kosztem
- znacznie lepsze komunikaty o błędach
- prosta składnia

Jak używać konceptów?

- aby **zwiększyć** abstrakcje w kodzie
- aby uogólniać algorytmy
- **używaj do wszystkich parametrów szablonu**
- używaj nazw konceptów zamiast `auto`

auto z konceptami

```
1  template<typename T>          //Why do you hate people?
2  void sort(T&);
3
4  template<typename T>          //Correct but "verbose"
5      requires Sortable<T>
6  void sort(T&);
7
8  template<Sortable T>          // Better
9  void sort(T&);
10
11 void sort(Sortable auto&);    // What we want :/
```

Praktyczne zastosowanie:

`https://github.com/cppfastio/fast_io`

Kiedy pisać własne koncepty?

- Jeżeli to tylko możliwe używaj konceptów (i ich logicznych kombinacji) z biblioteki standardowej.
- Unikaj pisania konceptów z tylko jedną własnością (Addable to słaby koncept, Number jest całkiem dobry)
- Umieszczenie niezwiązanych ze sobą operacji czy typów w jednym koncepcie to z reguły słaby pomysł

Źródła:

https://github.com/hniemeyer/IntroToConcepts/blob/master/IntroToConcepts.pdf
https://omnigoat.github.io/2020/01/19/cpp20-concepts/
https://en.cppreference.com/w/cpp/language/constraints
https://github.com/isocpp/CppCoreGuidelines/blob/master/CppCoreGuidelines.md