0.1 Definiowanie konceptów

Koncepty, takie jak `Equality_comparable` często można znaleźć w bibliotekach (np. w `The Ranges TS`), ale koncepty można też definiować samodzielnie:

```
template<typename T>
concept bool Equality_comparable = requires (T a, T b){
    { a == b } -> bool;
    { a != b } -> bool;
};
```

Koncept ten został zdefiniowany jako szablonowa zmienna. Typ musi dostarczać operacje `==` i `!=`, z których każda musi zwracać wartość `bool`, żeby być `Equality_comparable` . Wyrażenie `requires` pozwala na bezpośrednie wyrażenie jak typ może być użyty:

- `{ a == b }`, oznajmia, że dwie zmienne typu `T` powinny być porównywalne używając operatora `==`
- `{ a == b } -> bool` mówi że wynik takiego porównania musi być typu `bool`

Wyrażenie `requires` jest właściwie nigdy nie wykonywane. Zamiast tego kompilator patrzy na wymagania i zwraca `true` jeśli się skompilują a `false` jeśli nie. To bardzo potężne ułatwienie.

```
template<typename T>
concept bool Sequence = requires(T t) {
    typename Value_type<T>;
    typename Iterator_of<T>;

    { begin(t) } -> Iterator_of<T>;
    { end(t) } -> Iterator_of<T>;

    requires Input_iterator<Iterator_of<T>>;
    requires Same_type<Value_type<T>,
    Value_type<Iterator_of<T>>>;
};
```

Żeby być typu `Sequence`:

- typ `T` musi mieć dwa powiązane typy: `Value_type<T>` i `Iterator_of<T>`. Oba typy to zwykłe *aliasy szablonu*[1]. Podanie tych typów w wyrażeniu `requires` oznacza, że typ `T` musi je posiadać żeby być `Sequence`.

[1] nazwa odwołująca się do rodziny typów.

- typ `T` musi mieć operacje `begin()` i `end()`, które zwracają odpowiednie iteratory.
- odpowiedni iterator oznacza to, że typ iteratora typu `T` musi być typu `Input_iterator` i typ wartości typu `T` musi być taki sam jak jej wartość typu jej iteratora. `Input_iterator` i `Same_type` to koncepty z biblioteki.

Teraz w końcu możemy napisać koncept `Sortable`. Żeby typ był `Sortable`, powinien być sekwencją oferującą losowy dostęp i posiadać typ wartości, który wspiera porównania używające operatora `<`:

```
template<typename T>
concept bool Sortable = Sequence<T> &&
Random_access_iterator<Iterator_of<T>> &&
Less_than_comparable<Value_type<T>>;
```

`Random_access_iterator` i `Less_than_comparable` są zdefiniowane analogicznie do `Equality_comparable`

Często, wymagane są relacje pomiędzy konceptami. Np. koncept `Equality_comparable` jest zdefiniwoany by wymagał jeden typ. Można zdefiniować ten koncept by radził sobie z dwoma typami:

```
template<typename T, typename U>
concept bool Equality_comparable = requires(T a, U b){
    { a == b } -> bool;
    { a != b } -> bool;
    { b == a } -> bool;
    { b != a } -> bool;
};
```

To pozwala na porównywanie zmiennych typu `int` z `double` i `string` z `char*`, ale nie `int` z `string`.