1 Włączenie konceptów do standardu C++

Koncepty nie zostały włączone do standardu *C++17*. Krótkie uzasadnienie jest takie, że komisja nie osiągnęła porozumienia, że koncepty (określone w specyfikacji technicznej) osiągnęły wystarczające doświadczenie w zakresie wdrożenia i użytkowania, aby być wystarczające do dopuszczenia w obecnym projekcie. Zasadniczo komisja nie powiedziała "nie", ale "jeszcze nie".

Największe zastrzeżenia nie wynikały z problemów technicznych. Powstały następujące obawy:

- specyfikacja konceptów istniała w opublikowanej formie przez mniej niż cztery miesiące
- jedyna znana i dostępna publicznie implementacja konceptów znajduje się w nieopublikowanej wersji *kompilatora GCC*
- implementacja *kompilatora GCC* została opracowana przez tę samą osobę, która napisała specyfikację. W związku z tym implementacja jest dostępna do testowania, ale nie podjęto żadnej próby wprowadzenia w życia specyfikacji. A zatem specyfikacja nie jest przetestowana. Kilku członków komisji wskazało, że posiadanie implementacji wyprodukowanej ze specyfikacji ma decydujące znaczenie dla określenia kwestii specyfikacji.
- najbardziej znaczące i znane użycie konceptów jest dostępne w specyfikacji *Ranges TS*. Jest kilka innych projektów eksperymentujących z konceptami, ale żaden z nich nie zbliża się do skali, której można oczekiwać gdy programiści zaczną korzystać z tej funkcjonalności. Wydajność i problemy związane z obsługą błędów przy użyciu bieżącej implementacji GCC dowodzą, że nie wykonano większej próby używania konceptów.
- specyfikacja konceptów nie dostarcza żadnych definicji. Niektórzy członkowie komisji kwestionują użyteczność konceptów bez dostępności biblioteki definicji konceptów, takiej jak *Ranges TS*. Przyjęcie specyfikacji konceptów do *C++17* bez odpowiedniej biblioteki definicji niesie ryzyko zablokowania języka bez udowodnienia, że zawiera funkcje potrzebne do wdrożenia biblioteki, które mogłyby być zaprojektowane do konceptualizacji biblioteki standardowej.

Obawy techniczne:

- koncepty zawierają nową składnię do definiowania szablonów funkcji. Skrócona deklaracja szablonu funkcji wygląda podobnie to nieszablonowej deklaracji funkcji z wyjątkiem tego, że co najmniej jeden z jej

parametrów zostanie zadeklarowany ze specyfikatorem typu zastępczego `auto` albo nazwą konceptu. Obawa wynika z tego, że taka deklaracja:
`void f(X x){}`
definiuje nieszablonową funkcję jeśli `X` jest typem, ale definiuje szablon funkcji jeśli `X` jest konceptem. To ma subtelne konsekwencje dla tego czy funkcja może być zdefiniowana w pliku nagłówkowym, czy słowo kluczowe `typename` jest potrzebne by odnieść się do składowych typów typu `X`, czy istnieje dokładnie jedna zmienna lub brak lub kilka dla każdej deklarowanej zmiennej lokalnej, statycznej. itd.

- specyfikacja konceptów zawiera również składnię szablonów wstępnych, która pozwala ominąć rozwlekłą składnię deklaracji szablonu, do której wszyscy są przyzwyczajeni jednocześnie określając ograniczenia typu. Następujący przykład deklaruje szablon funkcji `f`, przyjmujący dwa parametry `A` i `B`, które spełniają wymagania konceptu `C`:
  `C{ A, B } void f(A a, B b);`
  Ta składnia nie jest lubiana. Wspomniano, że biblioteka *Ranges TS* jej używała w pewnych miejscach a grupa pracująca nad ewolucją biblioteki zażądała żeby ją zmienić i już nigdy nie używać.

- Są dwie formy definiowania konceptów: funkcja i zmienna. Forma funkcji istnieje po to by wspierać przeciążanie definicji konceptów oparte na parametrach szablonu. Forma zmiennej istnieje by wspierać nieco krótsze definicje:

```
//forma funkcji
template<typename T>
concept bool C(){
   return ...;
}

//forma zmiennej
template<typename T>
concept bool C = ...;
```

  Wszystkie koncepty, które można zdefiniować przy użyciu formy zmiennej można zdefiniować za pomocą formy funkcji. Stosowana forma wpływa na składnię wymaganą do oszacowania konceptu, a zatem użycie konceptu wymaga znajomości formy użytej do jego zdefiniowania. Wczesna wersja *Ranges TS* używała zarówno formy zmiennej, jak i funkcji do definiowania konceptów i niespójność spowodowała wiele błędów w specyfikacji. Aktualna *Ranges TS* wykorzystuje tylko formę funkcji do zdefiniowania określonych konceptów. Niektórzy członkowie komitetu uważają, że jedna forma definicji uprości język i uniknie trud-

ności w używaniu i nauczaniu. Zapewnienie odrębnej składni definicji konceptów, a nie określenie ich w kategoriach funkcji lub zmiennych uniknęłoby również dziwnej składni `concept bool`.

- została dodana możliwość używania `auto` jako specyfikatora dla parametrów szablonu bez typu:

  ```
  template<auto V>
  constexpr auto v = V*2;
  ```

  Z konceptami można by ograniczyć powyższy szablon tak, że typ `V` spełniałby wymagania konceptu `Integral`:
  ```
  template<Integral V>
  constexpr auto v = V*2;
  ```

  Jednak to jest ta sama składnia aktualnie używana przez *Concepts TS*, do deklarowania parametrów typu szablonu ograniczonego. Jeśli *Concepts TS* miały być wprowadzone, potrzebna by była inna składnia aby deklarować ograniczony parametr szablonu bez typu. Prawdopodobnie składnia stosowana przez *Concepts TS* bardziej nadaje się do deklarowania parametrów szablonów bez typu, jak pokazano powyżej, ponieważ pasuje do składni stosowanej dla innych deklaracji zmiennych. To oznacza, że nowa składnia do deklarowania ograniczonych parametrów typu byłaby pożądana ze względu na spójność języka.

- Koncepty były powszechnie oczekiwane w celu uzyskania lepszych komunikatów o błędach niż obecnie są generowane, gdy pojawiają się niepowodzenie podczas tworzenia szablonów. Teoria idzie, ponieważ koncepty pozwalają odrzucić kod oparty na ograniczeniu w punkcie użycia szablonu, kompilator może po prostu zgłosić błąd ograniczenia, a nie błąd w niektórych wyrażeniach w potencjalnie głęboko zagnieżdżonym stosie instancji szablonu. Niestety okazuje się, że nie jest tak proste, a używanie konceptów skutkuje gorszymi komunikatami o błędach. Niepowodzenia ograniczeń często pojawiają się jako błędy w przeciążeniu, co powoduje potencjalnie długą listę kandydatów, z których każda ma własną listę przyczyn odrzucenia. Identyfikacja kandydata, który był przeznaczony do danego użycia, a następnie określenie, dlaczego wystąpiło niepowodzenia ograniczeń, może być gorszym doświadczeniem niż nawigowanie w stosie tworzenia instancji szablonów.

- Wielu członków komisji wyraża zaniepokojenie faktem, czy obecny projekt konceptów wystarcza jako podstawa, na której można w przyszłości wdrożyć sprawdzenie pełnej definicji szablonu. Mimo zapewnień obrońców konceptów, że takie kontrole będą możliwe, wiele py-

tań pozostaje bez odpowiedzi, a członkowie komitetu pozostają bez przekonania. Wydaje się mało prawdopodobne, że obawy te zostaną rozwiązane w inny sposób niż poprzez wdrożenie sprawdzania definicji.

Wielu wierzy, że koncepty w jakiejś formie zostaną dodane do *C++19/20*.