C++20 – trzęsienie ziemi na koniec dekady

W sobotę, 15 lutego 2020, zakończyło się posiedzenie komisji standaryzacyjnej C++. Chwilę potem członkowie komisji poinformowali w serwisie Reddit [0], a także za pomocą innych kanałów komunikacji, że ostateczna wersja (szkicu) standardu (DIS – ang. Draft International Standard) jest gotowa do wysłania do tzw. National Bodies w celu poddania ostatecznemu głosowaniu, które to powinno być wyłącznie formalnością.

Cóż za zwieńczenie dekady[1]! C++20 – ponieważ tak będzie się nazywał nowy standard, jeśli wejdzie w życie w 2020 roku – będzie zawierał szereg rewolucyjnych i od dawna wyczekiwanych zmian. Nawet czytając same nagłówki sekcji tego artykułu, można zauważyć, że C++20 będzie wszystkim tym, czym miał być C++17 – i więcej! W opinii autora nadchodząca iteracja standardu wprowadza większą rewolucję niż osławiony C++11. Nie obejdzie się jednak bez łyżki dziegciu.

MODUŁY

Bez wątpienia jedną z najważniejszych zmian jest umożliwienie odejścia od archaicznego modelu dołączania zewnętrznego kodu poprzez dyrektywę preprocesora `#include`. Zamiast tego używane będą słowa `module` i `import`. Słowo `module` definiuje moduł, a `import` to instrukcja załączająca moduł.

Warto tutaj zaznaczyć, że nie są to nowe słowa kluczowe, tylko identyfikatory ze specjalnym znaczeniem (tak jak `override` i `final`). Dzięki temu zachowana zostaje kompatybilność z poprzednimi standardami, gdzie kod z Listingu 0 jest w pełni poprawny i taki pozostaje:

Listing 0. Poprawny kod C++17 pozostaje poprawny w C++20 [1]

```
class module{};

int main()
{
    module m;
}
```

Nie wszędzie jednak udało się zachować pełną kompatybilność wsteczną: Listingi 1 i 2 pokazują na przykładzie zmiennych globalnych kod, który wraz z C++20 przestaje się kompilować:

Listing 1. Poprawny kod C++17, niepoprawny w C++20 [2]

```
class module{};

module m;

int main()
{
}
```

Listing 2. Poprawny kod C++17, niepoprawny C++20 (w momencie pisania tego artykułu przyjmowany przez clang i gcc)

```
class module{};

namespace foo
{
    module m;
}

int main()
{
}
```

Choć z powyższego opisu mogłoby się wydawać, że jest to tylko formalna zmiana, to należy podkreślić, że odejście od preprocesora jest znaczącą innowacją. Preprocesor jest dziedzictwem języka C, a jego *design* pamięta lata 70. poprzedniego stulecia. Jego zasada działania jest bardzo prosta, ale jednocześnie problematyczna: dyrektywa `include` podmienia podaną nazwę lub ścieżkę do pliku na jego zawartość. Można to zaobserwować, dodając przełącznik `-E` do polecenia wywołania kompilatora gcc lub clang:

Listing 3. Zasada działania preprocesora

```
# cat x.cpp
#include "test"

int main()
{
        return foo();
}

# cat test
int foo()
{
        return 42;
}

# g++ x.cpp -E
# 1 "x.cpp"
# 1 "<built-in>"
# 1 "<command-line>"
# 1 "/usr/include/stdc-predef.h" 1 3 4
# 1 "<command-line>" 2
# 1 "x.cpp"
# 1 "test" 1

int foo()
{
 return 42;
}
# 2 "x.cpp" 2

int main()
{
 return foo();
}
```

1. Dekadą popularnie nazywa się okres od roku zakończonego 1 do roku zakończonego 0. Jest tak, ponieważ daty liczone są od roku zakończonego 1, a nie 0 (*notabene*, koncept zera dotarł do Europy dopiero w poprzednim millenium), tak więc pierwsza dekada to były lata [1–10], pierwsze stulecie to lata [1–100], a pierwsze millenium to lata [1–1000].