cppfront: rewolucyjny zwrot w historii języka C++?

We wrześniu br. na konferencji CppCon 2022 Herb Sutter, sekretarz komitetu standaryzacyjnego języka C++, wygłosił referat zatytułowany „Czy C++ może stać się 10 razy prostszy i bezpieczniejszy…?”, przedstawiając w nim pomysł na rozwiązanie problemów, z którymi C++ boryka się od kilkunastu lat. Chodzi tu o takie kwestie, jak stale rosnąca złożoność języka oraz brak wbudowanych mechanizmów gwarantujących bezpieczeństwo aplikacji. Czy zaproponowana idea oraz powiązany z nią eksperymentalny kompilator cppfront mogą stanowić rewolucyjny zwrot w historii języka C++? Niniejszy artykuł stanowi próbę znalezienia odpowiedzi na to pytanie.

PROBLEM

Zdecydowana większość doświadczonych programistów C++ na pytanie o to, jakie są największe problemy związane z tym językiem, wskaże najprawdopodobniej dwie kluczowe kwestie. Pierwszą z nich będzie niewątpliwe rosnąca z roku na rok złożoność samego C++ oraz otaczającego go ekosystemu. Proces narastania tej złożoności rozpoczął się kilkanaścia lat temu, mniej więcej w okresie wprowadzenia standardu C++11. Wynikał on z palącej potrzeby modernizacji języka z jednoczesnym zachowaniem kompatybilności wstecznej. Rezultat jest taki, że C++ rozrósł się bardzo zarówno w kontekście samej składni, jak i biblioteki standardowej. Kłopot jednak w tym, że projektanci języka nieraz musieli stawać na głowie, wymyślając skomplikowane rozwiązania dotyczące prostych problemów tylko po to, by pogodzić nowo dodawane właściwości z istniejącymi, często archaicznymi rozwiązaniami.

Świetnym przykładem jest tutaj mechanizm modułów wprowadzony w standardzie C++20, który próbuje zmodernizować istniejący – archaiczny, jak by nie patrzeć, mechanizm języka C++ służący do współdzielenia deklaracji oraz definicji pomiędzy jednostkami translacji, oparty na plikach nagłówkowych.

Patrząc na eleganckie rozwiązania tego problemu w językach takich jak Go czy Rust, widać, jak relatywnie prosto da się zaprojektować taki mechanizm, wychodząc z pozycji *carte blanche*. Każdy zaś, kto przyglądał się bliżej konstrukcji modułów wprowadzonych w standardzie C++20, na pewno zdaje sobie sprawę, z jaką ilością problemów i wyzwań musieli zmierzyć się jego projektanci. Ten poziom złożoności przenosi się automatycznie na użytkowników końcowych (w tym przypadku na „zwykłych” użytkowników języka C++). Skutkuje to powstawaniem takich inicjatyw jak *C++ Core Guidelines* (utrzymywany na bieżąco, obszerny zestaw wytycznych definiujących rekomendowany zakres stosowania języka oraz podzbiór jego bezpiecznych właściwości) czy też opracowań w stylu *Embracing Modern C++ Safely* (ponad 1300-stronicowa książka wydana pod koniec 2021 roku przez czterech inżynierów z firmy Bloomberg, przedstawiająca szczegółową analizę nowych właściwości C++ dodanych w standardach C++11 i C++14 w kontekście bezpieczeństwa i spójności przy utrzymywaniu dużych, wieloplatformowych projektów budowanych w oparciu o ten język). Dla jasności, zarówno projekt *C++ Core Guidelines*, jak i książka *Embracing Modern C++ Safely* stanowią nieocenione zasoby dla każdego profesjonalnego programisty C++, a jednocześnie monumentalny efekt ciężkiej pracy niemałej grupy bardzo zdolnych osób, którym niewątpliwie należy się szacunek i wdzięczność za wysiłek włożony w realizację tych opracowań. Niemniej jednak sam fakt, że ktoś musiał taką pracę wykonać, pokazuje, jak bardzo problem złożoności języka C++ wymknął się nam spod kontroli…

Co do drugiej dominującej problematycznej kwestii związanej z językiem C++ (oraz z językiem C, którego C++ stanowo przecież nadzbiór), tj. kwestii bezpieczeństwa, to chyba nie trzeba nikogo przekonywać o jej zasadności. Dane statystyczne dotyczące luk w oprogramowaniu (oraz liczone w miliardach dolarów straty z nimi związane) mówią tu same za siebie. Dość powiedzieć, że już nawet instytucje rządowe (takie jak np. amerykańska Agencja Bezpieczeństwa Narodowego) odradza stosowanie języków C oraz C++ przy budowaniu rozwiązań software'owych narażonych na potencjalne ataki.

Oczywistym jest, że te wszystkie, opisane powyżej niedogodności są ceną za zachowanie wstecznej kompatybilności z poprzednimi wersjami C++. Biorąc pod uwagę, jak gigantyczna jest istniejąca baza kodu napisanego w tym języku, utrata tej kompatybilności praktycznie nie wchodzi w grę. I w tym momencie koło się zamyka… Jak więc rozwiązać ten węzeł gordyjski?

PROPONOWANE ROZWIĄZANIE

To, że wsteczną kompatybilność języka C++ należy zachować, nie podlega dyskusji. To, że cena za to jest gigantyczna i sprawia, że C++ brnie w ślepą uliczkę, jest niezbitym faktem. Co w tej sytuacji można zrobić?

Pomysł na rozwikłanie tego dylematu przedstawił Herb Sutter na konferencji CppCon 2022, w ramach wspomnianej na początku tego artykułu prelekcji „Czy C++ może stać się 10 razy prostszy i bezpieczniejszy…?”. Ten prowokujący tytuł odnosi się niewątpliwie do opisanych wyżej problemów, zaś odpowiedź, którą przedstawił Herb, brzmi: tak, da się!

Pomysł jest genialny w swej prostocie. Polega on na tym, aby w odniesieniu do kwestii zachowania kompatybilności wstecznej zastosować takie samo podejście, jakiego używa się na porządku dziennym w stosunku do innych właściwości języka C++. Konkretnie chodzi tutaj o zasadę: *płacisz tylko za to, z czego korzystasz*.

Propozycja Herba Suttera jest następująca:

» Zaprojektujmy nową składnię C++ (nazywana dalej jako *składnia nr 2*), taką, która rozwiązuje najbardziej palące problemy