A Word About Modules

Gabriel Dos Reis
Microsoft

The C++ community has rightfully embraced "C++ Modules" as a language functionality to help the working programmer express software architecture boundaries, dependencies more formally. These structures take the celebrated One Definition Rule as foundational, as opposed to a property that needs to be re-discovered and to be checked over and over. Because of that, modules hold the promise of also improving compile-time, a critical productivity issue for modern C++. To get there, we need to take code hygiene seriously. To make a dent, a good module system for C++ cannot just be an embodiment of what is expedient in the moment.

At the Fall 2015 meeting in Kona, Hawai'i, the C++ Evolution Working Group took the necessary and courageous step of sending a design of modules for C++ to the C++ Core Working Group for review as basis for a Module Technical Specification. That design has no provision for exporting or importing macros across module boundaries. Note that, as ever, you can still use macros in any translation unit even if they are part of a module, as usual. It is a bold move to address fundamental problems at the core of software engineering with C++ and productivity tools support. A Technical Specification is designed as a vehicle to conduct bold experiments before we enshrine a feature into the standards text. For that to work, the TS needs to exist to serve as a basis of shared, common understanding for compilers, tool builders, and C++ users to experiment with. If the C++ committee is to consider modules for C++20, now is the time to act, now is the time to send the Modules TS draft for PDTS so we can learn what works in practice, what needs improvement, and from the experiments what would be effective ways to migrate from where we are today to where we would like to be in a world of C++ with modules. Further delays will harm the C++ community we serve.

To this date, there are ongoing implementations of the Module TS draft in at least three widely used C++ compilers. I am encouraged by the passion of the C++ community, the dedication of C++ compiler implementers, and the energy and feedback I get from the C++ community. We are facing a historic opportunity to fundamentally shape what it is to build software systems at scale with C++. But we are running out of time. We need to act. And learn from the experiment, to inform our decisions for the next steps. An experiment that is not just like how we have been writing systems the last four decades or so.

These are exciting time for C++ developers. Let's seize the opportunity to change how we write C++.