C++ Compile-time Reflection and Mock Objects

Gábor Márton, Zoltán Porkoláb

Department of Programming Languages and Compilers, Eötvös Loránd University

martongabesz@gmail.com
gsd@elte.hu

Reflection is an important tool in the hands of programmers since a while. Serializing objects, creating mock objects for testing or creating object relational mappings are just a few use cases. Writing generic code in Java or in Python for such use cases is possible today. Though, using reflection in these managed languages is doable only in runtime, therefore this implies runtime penalty. Currently C++ has a very limited capability of runtime reflection (operator typeid). [1]

ISO C++ started a study group (SG7) to examine the possibilities of compile-time reflection in C++. [2] With compile-time reflection it would be possible to have a generic library for serialization or for object relational mappings. There are several potential notions about how to approach this kind of reflection. For example introducing high-level new lingual elements like `static for`, or creating library interfaces which are hiding compiler intrinsics for each specific reflection subtask.

Without standardized C++ compile-time reflection, creating proxy objects or mock test objects is a repetitive and error-prone task. In this paper an alternative C++ compile-time reflection approach is discussed in favor of finding a generic solution for this task. The approach is based on introducing new library elements. Under the hood these library element implementations has to be compiler specific intrinsics (compiler specific expressions). With these expressions, variables and functions could be declared and defined from results of reflection queries.

References

[1] ISO International Standard ISO/IEC 14882:2011(E) Programming Language C++

[2] https://groups.google.com/a/isocpp.org/forum/#!forum/reflection