Orion: A Tool for Gathering Dependencies of C++ Source Files

Gabriel Dos Reis
Microsoft

Abstract

Migrating existing large code base to idiomatic contemporary ISO C++, or adopting C++ Modules, requires understanding the preprocessing dependencies of every source file. These dependencies include macros defined by the source file, macros undefined by the source file, macros tested by the source file, files included by the source file, pragmas defined by the source file, relative order of file inclusion, and other C preprocessor vagaries. For medium to large source code base, this is a gigantic task that cannot be tackled without efficient compiler-based tooling. Migration implies (semi-)automated rewrite. Most semantics-based rewrite and transformations require some form of AST from the compiler. However, the preprocessor dependencies can be done at the pre-parsing level. This is a necessary preliminary step to full AST-based transformation (refactoring).

Preprocessor-level dependencies

The analysis of a source file for preprocessor dependencies cannot just be a "regular" compiler invocation. For, in normal compile, the compiler is required to ignore parts of conditional compilation that are not taken. However, a good analysis tool must explore both taken and untaken branches of a conditional inclusion.

Given a source file, the tool will output:

- The source file path
- Include guard, if any, with source location information
- All macro definitions, along with source location information, and preprocessor conditions under which they are defined
- All macro undefinitions, along with source location information, and preprocessor conditions under which they are undefined
- All pragmas, along with source location information, and preprocessor conditions under which they are defined
- All file inclusions, along with source location information, and preprocessor conditions
- All (potential) names tested in a preprocessor condition, along with source location information
- For each file directly included via '#include', the directory where the nominated file was found, along with any conflicting search directories.

For a source file using a module, this output will also include

- Module names referenced in import declarations, along with source location information

It is important to note that the tool does not perform transitive inclusion of files. Rather it outputs raw, but faithful, information about the “high level human view” of the input source file. A separate utility is to combine the outputs to check for coherency of dependencies. The tool can be invoked with the usual compiler switches related to defining or undefining macros.

Output Format

The tool will output the information described in the previous section in a YAML format. The choice of YAML is largely a function of integration with other tools being developed in DevDiv. The output file can the be used as an input (one of many) to a refactoring tool or code analysis tool.

The tool as described in this document is a foundational infrastructure tool, and as such will not be directly manipulated