In-Source Mechanism to Identify Importable Headers

Document #:	P1905R0
Date:	2019-10-06
Project:	Programming Language C++
Audience:	SG-15, EWG

Reply-to: Corentin Jabot <corentin.jabot@gmail.com>

Target

C++20

Abstract

The standard specifies that *an importable header is a member of an implementation-defined set of headers*. This paper proposes a mechanism to specify that a header is importable from within the source code of that header.

Motivation

Defining the set of importable header is accepted to require the collaboration between

- Developers, as determining whether a header is importable (not affected by the state of the preprocessor), cannot be easily computed automatically.
- Build systems which are the primary tool by which developers can specify how a program can be compiled.
- The compiler which needs to know the list of importable header.

And of course, compiling a program often involves multiple build systems that do not share the same build file formats.

SG-15 is looking at recommending a set of formats and/or protocols that could be used to share whether a given header can be considered importable across these entities, however that presents several limitations:

- All tools involved need to be able to specify, share and consume that information which, given the state of the ecosystem, will take many years.
- It puts more knowledge about a program in the build system, and less in the source which makes maintenance more complicated and notably makes it harder to replace the build system.

- It makes it harder to develop and use zero-configuration build systems as well as header-only libraries - which would now need to be configured in build scripts so they can benefit from being importable.
- It put the decision of whether a library is importable on the build system maintainer rather than on the person who wrote the header - But only the people maintaining the code know whether a header is and will remain importable.

Proposal

We propose a `#pragma importable` which, when it appears at the beginning of a header indicates that the included header is importable.

This can be used:

- By compilers to treat the included header as an imported header unit (see 15.2.7)
- By build systems providing pre-scanning to pre-compile the importable header if the implementation supports that use case.
- By IDE and other tools to treat the header as importable which can have drastic performance benefits.

This attribute would not replace other implementation-defined mechanisms to specify that a header is importable, but rather add a new one.

[*Note:* Because the proposed mechanism is designed to be ignorable, it does not replace nor alleviate the need for include guards or `pragma once`. — *end note*]

Why a pragma?

The syntax chosen to identify importable headers mustn't make the program ill-formed if compiled with a C++17 compiler or one that does not understand that syntax. In general, whether an importable header is imported or included should not affect the behavior of a program.

This leave us with 2 choices:

- A pragma
- An attribute (eg. `[[importable]]`)

However, a pragma has a few benefits, most notably it can be handled in phase 4 and is already specified to appear at the beginning of a line, which makes it easier for tooling to deal with. It also matches the behavior of whether an include is treated as in import - decision handled during prepossessing.

Having a standard `#pragma` directive is novel, however, all existing implementation tested (gcc, clang, msvc, icc) correctly ignore a `#pragma importable` (none of the implementation give a meaning to `#pragma importable`), so the standard can safely claim that syntax.

It would ultimately be possible to use an attribute if there was a preference for that.

Proposed Wording

 Pragma directive [cpp.pragma]

A preprocessing directive of the form

```
# pragma importable
```

Indicates that the header or source file being processed and identified by its *header-name* is an importable header ([module.import]).

In the header or source file being processed, this directive shall not appear after a `#define` or `#include` preprocessing directive or any *preprocessing-token*.

[*Note:* It is implementation defined whether the header or source file being processed will be treated as if it were imported by an `import` directive ([cpp.import]) of the form:

import header-name ; new-line

— *end note*]

A preprocessing directive of the form

`# pragma` $_{opt}$*pp-tokens new-line*

causes the implementation to behave in an implementation-defined manner. The behavior might cause translation to fail or cause the translator or the resulting program to behave in a non-conforming manner. Any pragma that is not recognized by the implementation is ignored.

References

[N4830] Richard Smith *Working Draft, Standard for Programming Language C++* https://wg21.link/n4830