Proposal for C2x
WG14 N2563

Title:	Character encoding of diagnostic text
Author, affiliation:	Aaron Ballman
Date:	2020-09-09
Proposal category:	Change/Clarification Requests

Abstract: The standard is unclear and inconsistent regarding what character encoding should be used when issuing diagnostic text as a result of an operation with user-supplied text (`static_assert`, `#error`, etc).

Character encoding of diagnostic text

Reply-to: Aaron Ballman (aaron@aaronballman.com)
Document No: N2563
Date: 2020-09-09

Summary of Changes

N2563

- Original proposal

Introduction and Rationale

The standard provides a few mechanisms that suggest an implementation issues a diagnostic based on text written in the source code. However, the standard does not uniformly address what should happen if the execution character set of the compiler cannot represent the text in the source character set.

`[[deprecated]]`

The `[[deprecated]]` attribute specifies in its recommended practice:

> Implementations should use the deprecated attribute to produce a diagnostic message in case the program refers to a name or entity other than to declare it, after a declaration that specifies the attribute, when the reference to the name or entity is not within the context of a related deprecated entity. The diagnostic message may include text provided by the string literal within the attribute argument clause of any deprecated attribute applied to the name or entity.

`static_assert`

The `static_assert` declaration specifies, in part:

> Otherwise, the constraint is violated and the implementation shall produce a diagnostic message that includes the text of the string literal, if present, except that characters not in the basic source character set are not required to appear in the message.

`#error`

The `#error` directive specifies, in part:

> …causes the implementation to produce a diagnostic message that includes the specified sequence of preprocessing tokens.

`[[nodiscard]]`

N2448 added an optional string literal argument to the `[[nodiscard]]` attribute similar to the one for the `[[deprecated]]` attribute in that the text is recommended to appear in a diagnostic message. The proposal specifies:

> The diagnostic message may include text provided by the string literal within the attribute argument clause of any nodiscard attribute applied to the name or entity.

This proposal was adopted with the understanding that the character encoding issue would be resolved in a future paper.

Proposal

To understand the proposal, we have to first understand something about character sets. The source character set is the character set used to encode the input to the implementation (and is often determined by the encoding the user's editor saved the source code file in). The execution character set is the character set used by the target architecture the implementation is translating the source code for. Neither character set describes the environment in which diagnostics are displayed to the user. For instance, the user may encode their source in UTF-16, be compiling their code for a system whose execution character set is UTF-8, but the shell running the compiler may be using Windows Latin 1. There is no relationship between the character set used to display diagnostics from the implementation and any of the character sets defined by the C Standard. Further, there are no requirements on what producing diagnostics actually means – even if the user pipes all compiler diagnostic output to /dev/null, the implementation is still strictly conforming. This freedom allows an implementation to provide the correct behavior for their diagnostic output environment (which could be a shell, a file, a serial interface, or any number of other things the standard may or may not be able to talk about).

Because the nature of diagnostic messages is wholly in the realm of Quality of Implementation, the proposal is to place no character set related requirements on the diagnostic output with the understanding that implementations will do what makes the most sense for their situation when issuing diagnostics in terms of which characters need to be escaped or otherwise handled in a special way. However, the proposal does clean up some of the wording around the diagnostic-producing constructs to ensure they give the proper recommendations.

Proposed Wording

The wording proposed is a diff from the committee draft of WG14 N2478 with WG14 N2448 applied. Green text is new text, while ~~red~~ text is deleted text.

Modify 6.7.10p3:

The constant expression shall be an integer constant expression. If the value of the constant expression compares unequal to 0, the declaration has no effect. Otherwise, the constraint is violated and the implementation shall produce a diagnostic message ~~that~~ which should include~~s~~ the text of the string literal, if present~~, except that characters not in the basic source character set are not required to appear in the message~~.

Modify 6.7.11.2p4 (added by WG14 N2448):

The diagnostic message ~~may~~should include text provided by the string literal within the attribute argument clause of any `nodiscard` attribute applied to the name or entity.

Modify 6.7.11.4p5:

Implementations should use the `deprecated` attribute to produce a diagnostic message in case the program refers to a name or entity other than to declare it, after a declaration that specifies the attribute, when the reference to the name or entity is not within the context of a related deprecated entity. The diagnostic message ~~may~~should include text provided by the string literal within the attribute argument clause of any `deprecated` attribute applied to the name or entity.

Acknowledgements

I would like to recognize the following people for their help in this work: Tom Honermann, Hubert Tong, Joseph Myers, Alisdair Meredith, Steve Downey, JF Bastien, and Martinho Fernandes.