Document Number: J16/04-0067 = WG21 N1627
Date: 2004-04-04
Reply To: William M. Miller
The MathWorks, Inc.
`wmm@world.std.com`

"Conditionally-Supported Behavior" (Rev. 1)

I. Introduction

The previous version of this document (J16/04-0004 = WG21 N1564) proposed a new conformance category, "conditionally-supported behavior," applying to features an implementation could choose to support or not. If supported, the feature would be treated as implementation-defined; otherwise, the implementation would be required to issue a diagnostic upon encountering that feature.

As this concept was discussed by the Core Language Working Group at the Sydney (March, 2004) meeting, the group agreed that the original paper should be modified in several ways. Most importantly, the group decided that the concept should be expanded in order to permit its application to additional language and library features beyond those originally envisioned. In particular, it is anticipated that the library may be augmented with packages that are optional but, if present, must be implemented as specified in the Standard (file-system support was suggested as one example). The previous implication of "implementation-defined behavior" is, obviously, inappropriate for such features.

In addition, the CWG felt that several of the specifications that were suggested as candidates for "conditionally-supported behavior" might in fact be more appropriately recategorized as ill-formed, requiring a diagnostic. The proposed changes from the previous version of this document falling into this category include those for 2.1.3.1, 2.13.2, and 5.2.2.

A final influence was the fact that there is currently an effort to synchronize the specification of the C++ preprocessor with that of C99. The concept of "conditionally-supported behavior" is being presented to WG14 for their consideration, so it was decided to defer action on all preprocessor-related features until it is clear what the C Committee's decision will be. Edits from the previous version of this document affected by this consideration include 2.1 (both changes), 2.13.4, 2.4, 2.8, and all changes in clause 16.

The following edits are those from the earlier paper that were not deferred for either of the preceding reasons, modified to reflect the new definition of "conditionally-supported behavior."

II. Additions and Changes

The following citations are all relative to the wording and numbering of the 2003 version of the Standard.

1.3: Add the following as 1.3.2 and renumber all following definitions accordingly. *[Drafting note: cross-references within the following are to the current section numbers.]*

> **1.3.2 conditionally-supported behavior**
> behavior evoked by a program construct that is not a mandatory requirement of this International Standard. If a given implementation *supports* the construct, the behavior shall be as described herein; otherwise, the implementation shall document that the construct is not supported and shall treat a program containing an occurrence of the construct as ill-formed (1.3.4).

1.4¶1: Add the indicated words:

> The set of *diagnosable rules* consists of all syntactic and semantic rules in this International Standard except for those rules containing an explicit notation that "no diagnostic is required" or which are described as resulting in "undefined behavior." **In addition, an occurrence of a program construct described herein as "conditionally-supported" or as resulting in "conditionally-supported behavior" when the implementation does not in fact support that construct shall also be deemed a violation of a diagnosable rule.**

5.2.10: Add the following as a new paragraph 8, renumbering the following paragraphs:

> Converting a pointer to a function to a pointer to an object type or vice versa evokes conditionally-supported behavior. In any such conversion supported by an implementation, converting from an rvalue of one type to the other and back (possibly with different cv-qualification) shall yield the original pointer value; mappings between pointers to functions and pointers to objects are otherwise implementation-defined.

7.4¶1: Change as indicated:

> ~~The meaning of an~~ **An** `asm` declaration **evokes conditionally-supported behavior. If supported, its meaning** is implementation-defined.

7.5¶2: Change as indicated:

> The *string-literal* indicates the required language linkage. ~~The meaning of the *string-literal* is implementation-defined. A *linkage-specification* with a string that is unknown to the implementation is ill-formed.~~ **This International Standard specifies the semantics of C and C++ language linkage. Other values of the *string-literal* evoke conditionally-supported behavior, with implementation-defined semantics. [*Note:* Therefore, a *linkage-specification* with a *string-literal* that is unknown to the implementation requires a diagnostic.** ~~When the *string-literal* in a *linkage-specification* names a programming language, the spelling of the programming language's name is implementation-defined. *[Note:*~~ **It** is

recommended that the spelling be taken from the document defining that language, for example `Ada` (not `ADA`) and `Fortran` or `FORTRAN` (depending on the vintage). ~~The semantics of a language linkage other than C++ or C are implementation-defined.~~]

14¶4: Change as indicated:

A template, a template explicit specialization (14.7.3), or a class template partial specialization shall not have C linkage. If the linkage of one of these is something other than C or C++, the ~~behavior is implementation-defined~~ **result is conditionally-supported behavior, with implementation-defined semantics**.