Doc. no.: X3J16/96-0174
WG21/N0992
Date: September 24, 1996
Project: Programming Language C++
Reply to: William M. Miller
wmm@world.std.com

Defining Conformance

I. Introduction

The conformance model for C++ implementations and programs is defined in subclauses 1.3 **[intro.compliance]** and 1.4 **[intro.defs]**. Together, these subclauses define categories of programs (well-formed, ill-formed, having undefined behavior) and the requirements placed on implementations when presented with these various kinds of program input. As it currently stands, however, the text of these subclauses is in need of improvement.

II. Analysis

There are at least two problematic aspects to the treatment of conformance in the current Working Paper. The first, and most serious, is that it is inconsistent, both internally and with the treatment of program errors elsewhere in the WP. Second, it effectively forbids implementations to use more aggressive detection and reporting of program errors than the minimal approach required by the WP.

II. A. Inconsistency

The problem of inconsistency arises primarily because the same terms or rules are described in multiple locations and exceptions to the general form are made in one place but not another. For instance, 1.4 defines an "ill-formed program" as

> input to a C++ implementation that is not a well-formed program (*q.v.*)

and a "well-formed program" as

> a C++ program constructed according to the syntax rules, diagnosable semantic rules, and the One Definition Rule (3.2).

Thus, according to this pair of definitions, to qualify as "ill-formed" a program must contain a violation of the ODR, a syntax rule, or a diagnosable semantic rule. However, 1.3¶5 contains a much broader definition of "ill-formed:"

> Whenever this International Standard places a requirement on the form of a program (that is, the characters, tokens, syntactic elements, and types that make up

> the program), and a program does not meet that requirement, the program is ill-formed...

Which is the correct definition of "ill-formed?"

Similarly, there are inconsistencies in the requirements placed on implementations when presented with ill-formed program input. The just-cited passage in 1.3¶5 requires that all ill-formed programs be diagnosed:

> [When] a program does not meet that requirement, the program is ill-formed and the implementation shall issue a diagnostic message when processing that program.

However, 1.3¶3 allows an implementation not to diagnose certain ill-formed programs:

> If an ill-formed program contains no violations of diagnosable semantic rules, this International Standard places no requirement on implementations with respect to that program.[1]

In addition to internal inconsistencies within these two subclauses, there are also contradictions with other parts of the WP. For instance, subclauses 1.3 and 1.4 are written with the assumption that all syntax rules are diagnosable; e.g., 1.3¶2:

> Every conforming C++ implementation... shall issue at least one diagnostic message when presented with any ill-formed program that contains a violation of any diagnosable semantic rule or of any syntax rule.

However, there are a number of syntactic rules that that do not require diagnostics, either because they are explicitly so annotated (2.7¶1, non-whitespace characters following a form feed or vertical tab in a // comment; 2.10¶2, identifiers containing double underscore or beginning with an underscore and a capital letter) or because they result in undefined behavior (2.13.2¶3, use of an undefined escape sequence).

II. B. Diagnostic Restrictions

According to 1.3¶2,

> Every conforming C++ implementation shall, within its resource limits, accept and correctly execute well-formed C++ programs.

The set of "well-formed C++ programs" includes programs containing only errors for which no diagnostic is required. Even if an implementation is capable of detecting some

[1] It should also be noted in passing that a program containing only syntax errors – "ill-formed" but "[containing] no violations of diagnosable semantic rules" – does not require an implementation to issue a diagnostic message according to this paragraph, even though the preceding paragraph states that such a message *is* required.

of these errors, it is not permitted to issue a diagnostic and refuse to create an executable; instead, it must "accept and correctly execute" them. In fact, an implementation would be rendered nonconforming if it were incapable of producing an executable in the presence of certain of these errors.[2]

II. C. Rationale for Proposed Solution

One possible approach to rectifying the problems identified above would be to edit the individual offending statements on a piecemeal basis. This editing would primarily take the form of inserting additional qualifications into overly broad existing verbiage.

This approach, however, would result in complicating still further the already complex existing specification. This author, at least, believes that the current problems result mainly from a complex set of overlapping categories and the failure to clearly specify how they relate to each other. For instance, 1.3¶5 classifies rules into "form rules" and "execution rules," while 1.3¶1-2 classifies them into syntactic rules, diagnosable semantic rules, and semantic rules for which no diagnostic is required. Each is a reasonable taxonomy, but confusion results when each is used independently to try to specify the requirements on implementations. Program text is classified as well-formed, diagnosably ill-formed, and otherwise ill-formed; these categories do not correspond exactly with either of the rules taxonomies (the One Definition Rule is a semantic rule for which no diagnostic is required; violations of the ODR render a program ill-formed, unlike violations of other no-diagnostic-required semantic rules), and this classification is also used independently to try to specify implementation requirements, resulting in yet more confusion.

The approach advocated in the proposal below takes the opposite direction, simplifying the description by categorizing programs and rules in exact correlation with the requirements on implementations. Implementations are required to diagnose programs with violations of diagnosable rules; such programs are called "ill-formed."[3] Implementations are required to accept and correctly execute those programs that have no violations of rules and whose execution-time data meets the Standard's constraints; such programs and data are called "correct." All other programs and data result in undefined behavior, and the implementation is unconstrained at both compilation and execution time when processing such programs.

---

[2] One such scenario might be a failure of the program to define an otherwise-unused virtual function. According to 3.2¶2, no diagnostic is required for an undefined virtual function that is neither called nor used to form a pointer-to-member, so a program with such an error is "well-formed." The widely-used "vtable" implementation technique, however, might well result in a linkage error, thus preventing the implementation from "accepting and correctly executing" a well-formed program.

[3] Although achieving this identification between "diagnostic required" and "ill-formed program" was not the primary motivation for this proposal, the traffic on the email reflectors during its discussion has indicated strong sentiment in favor of making such an identification.

This proposal renders oxymoronic the formulation, "ill-formed, no diagnostic required." However, only a small number of locations in the WP currently employ this formulation, and they are all individually corrected below in the detailed proposal.

In addition, the proposal makes a few unrelated changes to subclause 1.3. 1.3¶6 appears to deal more with the conventions employed in the body of the Standard than with implementation compliance, so the proposal suggests moving it to the end of 1.1. Also, the wording was changed in a couple of places to clarify that additional libraries are acceptable extensions, and that it is permissible for extensions to change unspecified or implementation-defined behavior.

III. Detailed Proposal

A. Move the text of 1.3¶6 to the end of subclause 1.1.

B. Replace the entire subclause 1.3 with the following:

> 1.3 Implementation Compliance
>
> Although this International Standard states only requirements on C++ implementations, those requirements are often easier to understand if they are phrased as requirements on programs, parts of programs, or execution of programs. Except for those rules for which the Standard explicitly states, "No diagnostic is required," violations of the syntactic or semantic rules in this Standard are required to be diagnosed. Rules for which a diagnostic is required are called *diagnosable rules.*
>
> Program text that violates one or more diagnosable rules is called *ill-formed*; conversely, a program that contains no such violations is called *well-formed.* A conforming C++ implementation shall issue at least one diagnostic message when processing ill-formed program text.
>
> Well-formed programs containing constructs that violate rules for which no diagnostic is required, or whose behavior is not described by this International Standard, are said to exhibit *undefined behavior*. A program containing no such constructs is called a *correct program.* Undefined behavior also occurs when the values of execution-time data upon which a correct program operates violate constraints of the Standard or when the results of a program's operation on a value are not described by the Standard. Data which do not cause undefined behavior are called *correct data.* No requirements are placed on a conforming C++ implementation with respect to programs with undefined behavior.
>
> A conforming C++ implementation shall, within its resource limits, accept and correctly execute (1.8) any correct program operating on correct data.

Two kinds of implementations are defined: *hosted* and *freestanding*. For a hosted implementation, this International Standard defines the minimum set of available libraries. A freestanding implementation is one in which execution may take place without the benefit of an operating system, and has an implementation-defined set of libraries that includes certain language-support libraries (17.3.1.3).

A conforming implementation may have extensions (including additional library functions, classes, etc.), provided they do not alter the specified behavior of any correct program operating on correct data. One example of such an extension is allowing identifiers to contain characters outside the basic source character set. Implementations are required to diagnose programs that are ill-formed because of the use of such extensions. Having done so, however, they can process and execute such programs.

C. Add or replace the following entries in subclause 1.4:

correct data: Data whose values produce specified, implementation-defined, or unspecified behavior when used in a given correct program (q.v.).

correct program: A well-formed program (q.v.) containing no violations of rules for which no diagnostic is required and whose constructs do not engender undefined behavior (q.v.).

implementation-defined behavior: Behavior, for a correct program construct and correct data, that depends on the implementation and that each conforming implementation shall document.

undefined behavior: Behavior by an implementation or program for which the Standard imposes no requirements. Such behavior results when a well-formed program (q.v.) violates one or more rules for which no diagnostic is required, when the data upon which it operates violate constraints on their values, or when the Standard does not explicitly describe the behavior of a construct or data value.

well-formed program: A C++ program containing no violations of diagnosable rules.

D. In 3.3.6¶1, replace the phrase “the program’s behavior is ill-formed, no diagnostic is required” with “the program’s behavior is undefined.”

E. In 12.8¶4, replace the phrase “any use of X’s copy constructor is ill-formed because of the ambiguity; no diagnostic is required” with “any use of X’s copy constructor results in undefined behavior because of the ambiguity.”