Document Number: J16/04-0167 = WG21 N1727
Date: 2004-11-08
Reply to: William M. Miller
Edison Design Group, Inc.
wmm@edg.com

Changing Undefined Behavior into Diagnosable Errors

I. Background

Document J16/04-0004 = WG21 N1564 proposed a new conformance category, called "conditionally-supported behavior," to supplement the existing categories of undefined behavior, implementation-defined behavior, ill-formed programs, and well-formed programs. As originally envisioned, this category would require that various program constructs either be diagnosed as not supported by the implementation or be treated as implementation-defined behavior.

At the Sydney (March, 2004) meeting, the Core Language Working Group discussed this paper and adopted several changes in direction, described in document J16/04-0067 = WG21 N1627. In particular, the CWG felt that three of the items that were originally proposed for conditionally-supported behavior should actually become ill-formed, requiring a diagnostic. The purpose of this paper is to provide a stimulus and nexus for further discussion of those changes.

In each of the cases listed below, the proposed change (*mutatis mutandis*) is to replace "the behavior is undefined" with "the program is ill-formed." As noted in 1.4¶8, implementations are free to provide extensions in these areas (presumed to be the original rationale for undefined behavior in these particular cases), provided that they issue a diagnostic when such an extension is used.

II. Integer Literals

According to 2.13.1¶2,

> The type of an integer literal depends on its form, value, and suffix. If it is decimal and has no suffix, it has the first of these types in which its value can be represented: `int`, `long int`; if the value cannot be represented as a `long int`, the behavior is undefined.

The CWG felt that undefined behavior in this instance was inappropriate and would prefer to require implementations to issue a diagnostic. (Note that this change overlaps and should be incorporated into the proposed addition of the `long long` and `unsigned long long` types; see paper J16/04-0005 = WG21 N1565 and successors.)

III. Character Escapes

The current wording of 2.13.2¶3 states,

> If the character following a backslash is not one of those specified, the behavior is

> undefined.

Again, the CWG felt that a required diagnostic would be more appropriate.

IV. Passing Non-POD Objects to Ellipsis

The existing wording (5.2.2¶7) makes it undefined behavior to pass a non-POD object to an ellipsis in a function call:

> When there is no parameter for a given argument, the argument is passed in such a way that the receiving function can obtain the value of the argument by invoking `va_arg` (18.7). The lvalue-to-rvalue (4.1), array-to-pointer (4.2), and function-to-pointer (4.3) standard conversions are performed on the argument expression. After these conversions, if the argument does not have arithmetic, enumeration, pointer, pointer to member, or class type, the program is ill-formed. If the argument has a non-POD class type (clause 9), the behavior is undefined.

Once again, the CWG saw no reason not to require implementations to issue a diagnostic in such cases.