Document: N1531
Date: 2010/11/03
Author: Jim Thomas and Fred Tydeman
Subject: Clarifications for wide evaluation

6.3 #1 says the clause covers both implicit conversions and explicit conversions (casts).

Implicit conversions include the usual arithmetic conversions, which are not required to remove extra range or precision. Implicit conversion might also be considered to include assignments (which store in an object of the type, hence remove extra range and precision), argument passing (which assigns the argument values to the parameters, hence removes extra range and precision), and function return (which Annex F requires to remove extra range and precision).

Explicit conversions (casts) are intended to remove extra range and precision. For example,

158) The **`return`** statement is not an assignment. The overlap restriction of subclause 6.5.16.1 does notapply to the case of function return. The representation of floating-point values may have wider range or precision than implied by the type; a cast may be used to remove this extra range and precision.

Clause 6.3.1.4 and 6.3.1.5 define conversions involving real floating types, without distinguishing between implicit and explicit conversions, or saying whether or when extra range and precision are removed. Different places in the draft seem to make different assumptions about what these clauses require..

6.3.1.8 (Usual arithmetic conversions) refers to conversions of operands of a floating type to another floating type. It says

The values of floating operands and of the results of floating expressions may be represented in greater precision and range than that required by the type; the types are not changed thereby.63)

Thus the conversions called for by 6.3.1.8, presumably those defined above in 6.3.1.4 and 6.3.1.5, are not required to remove extra range and precision. (Requiring it would be inconsistent with wide evaluation.)

But the footnote assumes 6.3.1.4 and 6.3.1.5 do require removing extra range and precision. It says

63) The cast and assignment operators are still required to perform their specified conversions as described in 6.3.1.4 and 6.3.1.5.

The following changes are intended to clarify when the result of a conversion can have extra range and precision.

6.3.1.4 #2 (integer to floating conversion): Append

Results of some implicit conversions (6.3.1.8, 6.8.6.4) may be represented with extra range and precision.

6.3.1.5 #1, #2 change

When a **float** is promoted to **double** or **long double**, or a **double** is promoted to **long double**, its value is unchanged (if the source value is represented in the precision and range of its type).

When a **double** is demoted to **float**, a **long double** is demoted to **double** or **float**, or a value being represented in greater precision and range than required by its semantic type (see 6.3.1.8) is explicitly converted (including to its own type), if the value being converted can be represented exactly in the new type, it is unchanged. If the value being converted is in the range of values that can be represented but cannot be represented exactly, the result is either the nearest higher or nearest lower representable value, chosen in an implementation-defined manner. If the value being converted is outside the range of values that can be represented, the behavior is undefined.

to

When a value of real floating-point type is converted to a real floating type, if the value being converted can be represented exactly in the new type, it is unchanged. If the value being converted is in the range of values that can be represented but cannot be represented exactly, the result is either the nearest higher or nearest lower representable value, chosen in an implementation-defined manner. If the value being converted is outside the range of values that can be represented, the behavior is undefined. Results of some implicit conversions (6.3.1.8, 6.8.6.4) may be represented with extra range and precision.

6.3.1.8, footnote 63: change

63) The cast and assignment operators are still required to perform their specified conversions as described in 6.3.1.4 and 6.3.1.5.

to

63) The cast and assignment operators are still required to remove extra range and precision.

6.5.4 (Cast operator) #6 To the current sentence

If the value of the expression is represented with greater precision or range than required by the type named by the cast (6.3.1.8), then the cast specifies a conversion even if the type of the expression is the same as the named type.

append “and removes any extra range and precision.”

[Why is the reference to 6.3.1.8 here?]

In addition to casts, assignments and argument passing are required to remove extra range and precision. At the last meeting we agree that, for Annex F implementations, function returns too

would be required to remove extra range and precision. Currently, Annex F says the return "expression is converted to the return type of the function." This is not sufficient because conversions include implicit conversions that do not remove extra range and precision.

F.6 (The return statement): Change

If the return expression is evaluated in a floating-point format different from the return type, the expression is converted to the return type of the function and the resulting value is returned to the caller.

to

If a function has floating return type, the return expression is converted as if by assignment to the type of the function and the resulting value is returned to the caller.