**Document**: N1472
**Date**: 2010/05/10
**Author**: Jim Thomas
**Subject**: Comparison Macro Argument Conversion

7.12.14 specifies function-like macros for comparing real-floating arguments. An example is isgreater(real-floating x, real-floating y). Its Description says “isgreater(x, y) is always equal to (x) > (y); however, unlike (x) > (y), isgreater(x, y) does not raise the “invalid” floating-pint exception when x and y are unordered.” The part after the semicolon isn’t relevant here.

QUESTION: Are the arguments, which may be evaluated in a wider format, first converted to their type before the relationship is determined? For example, if a wide evaluation mode is in effect, on IEEE and most other floating-point systems, if evaluation is to double or wider, then isgreater(4.0f + FLT_EPSILON, 4.0f) is true if arguments are not converted to their type, but false if they are (assuming default rounding mode). Thus, the question of whether widened arguments are narrowed matters to program behavior. Clarification in the standard would be helpful.

The conversion of arguments to their type is explicitly stated for the classification macros in 7.12.3, but this clarification does not appear for 7.12.14.

The idea of the comparison macros being like relational operators and the Description statements like “isgreater(x, y) is always equal to (x) > (y)” might suggest that widened arguments are not narrowed, because widened operands of relational operators are not narrowed.

On the other hand, implementation techniques that determine an intrinsic function for the macro expansion based on the type of the arguments will naturally narrow widened arguments. Such is the case with the tgmath mechanism and with mechanisms based on sizeof or typeof or such. Some (most? all?) implementations (with wide evaluation modes) use such techniques and narrow widened arguments.

One can argue that statements like “isgreater(x, y) is always equal to (x) > (y)” do not settle the issue, because the x and y should be taken to be variables, not general expressions, and variables cannot be widened.

Of course the careful programmer can assure that arguments are narrowed by casting them to their type. If the implementation uses a standard evaluation method, i.e., FLT_EVAL_METHOD >= 0, the even more careful programmer can assure that arguments are not narrowed by casting arguments that are expressions subject to wide evaluation to their evaluation format (float_t or double_t).

Alternatives:

1. Specify that widened arguments of comparison macros must be narrowed (like classification macros).
2. Specify that widened arguments of comparison macros must not be narrowed (like relational operators).
3. Clarify that whether widened arguments of comparison macros are narrowed is unspecified.

If it were 12 years ago, I’d prefer 2. Now, if all implementations (with wide evaluation) narrow the arguments, then I’d prefer 1. The unhappy compromise in alternative 3 might be improved slightly by deprecating one of the behavior alternatives.