Doc. No.: WG14 N1228
Date: 2007/03/26
Project: Programming Language C (TR 24732)
Author: Raymond Mak (rmak@ca.ibm.com)
Subject: Lack of a portable way to code fpclassify

Background

The purpose of this document is to urge the Committee to take a look if a portable solution to distinguish decimal types from float/double/long double can be provided.

With the progress in the work of the Decimal Floating Pointer Technical Report (TR24732), there are early implementers working on adding decimal types. The work in gcc is an example. The experience from these implementers have shown that there is a need for a portable way to code fpclassify as well as other classification macros in C99 7.12.3. We understand that the committee had discussed this issue in the past, and that it felt in general the implementation has already been given sufficient freedom to provide a technical solution. In view of the experience from gcc, we would like to ask the committee to take a second look to see if a portable solution is possible.

Discussion

Even though the Standard considers an implementation of C99 consists of both the compiler and the runtime library, and the two should not be separated when we talk about an implementation, in practice, we often find more than one compilers exist on a platform. Different compilers usually use the same underlying runtime, which is shipped with the operating system (e.g. libc on UNIX systems). That is, we think of the runtime library as part of the operating environment. This is reinforced by the fact that the compiler and library are often versioned differently, and maintained on different service streams. Therefore library writers are conscious about the code that is put into the system headers. The code may favor a particular compiler (the default one on that platform, gcc on Linux, for example), giving better performance if that compiler is used, a fallback code path is often put in to catch the case when another compiler is used.

C99 provides a suggestion for coding fpclassify as an example in 7.12.3.1. Even though this may not be the most efficient solution for all compilers, this is at least a portable way to achieve the functionality, and can be used as a fallback path in the system header. However, with the introduction of decimal types, the sample code no longer works. It would be preferable if the Standard could provide a revised suggestion, along the spirit of the example in 7.12.3.1, on how to handle decimal types.

An attempt to a solution

There appears to be no C99 conforming solution to distinguish float/double/long double from the new decimal types. In the special case when the former is a binary floating type, and the non-standard ***`typeof`*** operator is used, the following solution is plausible.

typeof is introduced in gcc. This operator returns the type of the operand; the result can be used as a type name. Refer to the gcc documentation section 5.6 for details (http://gcc.gnu.org/onlinedocs/gcc-4.1.2/gcc.pdf).

Using ***typeof***, we can write a macro as follows:

```
#define  is_dfp(x)  (0.3DL == (_Decimal128)(typeof(x)) 0.3DL)
```

The macro returns 1 if the type of x is _Decimal32/64/128; and returns 0 if it is a binary floating type with a mantissa precision of less than or equal to about 106 bits.

The left-hand side of the comparison is exact. The righ-hand side goes through a conversion, which is exact if x is _Decimal32/64/128. The binary expansion of 0.3 is

0.01001100110011001100110011 …

If x is a binary floating type, we want to show that the rounding error in the two conversions cannot cancel out each other, and so the comparison would always fail.

If x is a binary floating type, the first cast on the right-hand side would incur a rounding error of at least 0.001100110011…ULP. (Note: This is the string of bits get dropped (round down), or the difference between 1 and 0.1100110011 … (round up). If we start dropping at other bit positions, the value would be larger.) This value is 1/5ULP. The ULP for 0.3 when represented in a w bits mantissa is at least 0x1p(-2-w). It can be larger if a hidden bit is not used in the representation, or if the number is denormalized. Setting w to 106 gives an error magnitude of at least 6.16E-34 for the first conversion.

The result of this first cast is one of two values bounding the true value of 0.3. The second cast takes one of these values and converts to _Decimal128. The error incurred is at most 3.0E-34. (1ULP of 0.3 in _Decimal128.) The error in the second cast is therefore too small to compensate for the first one, regardless of the combination of rounding modes in the two casts. The comparison would return false.

Even though this works for most of the existing implementations for binary generic floating types, this is not an elegant solution. In addition, it requires the use of a non-C99 ***typeof*** operator. It would be preferable if a general solution is possible.

Conclusion

The above attempt for a solution shows that it is difficult, if not impossible, to distinguish the two real floating types. Even though typeof is supported by a number of compilers, it is not C99 conforming. And the suggested expression does not work for all generic floating types. If a macro like the above is what we need, maybe a cleaner alternative is to define is_dfp as an operator in the language.