A Proposal to add max significant decimal digits macros to the C Standard Library.

Document number: JTC 1/SC22/WG14/N1151
Date: 2005-11-30, version 1
Project: Languages C++
References: C++ ISO/IEC IS 14882:1998(E),

A Proposal to add a max significant decimal digits value
to the C++ Standard Library Numeric limits, Paul A Bristow
Document number: JTC 1/SC22/WG21/N1822=05-0082
http://www2.open-std.org/JTC1/SC22/WG21/docs/papers/2005/n1822.pdf,

William Kahan http://http.cs.berkley.edu/~wkahan/ieee754status/ieee754.ps

Reply to: Paul A Bristow, pbristow@hetp.u-net.com, J16/04-0108

Introduction

Following favourable comment on my proposal above to add to the C++ Standard Library, I think it would be rational to add equivalent macros to the C equivalent. These values could of course be used by C++ implementations of std::numeric_limits, as well as providing an equivalent in purely C programs.

The case for the these values has been discussed in detail in the above paper, but a brief summary follows.

C99[ISO:9899] provides numeric limits 18.2.1 including

```
numeric_limits<Floating-Point Type>::digits10
```

also available (and often implemented using) via C macros FLT_DIG, DBL_DIG, LDBL_DIG.

The macro stores the number of decimal digits that the type can represent without change.

In effect, it is the number of decimal digits GUARANTEED to be correct (after rounding).

While useful, this does not provide another value, often more useful, the number of potentially significant decimal digits that the type can represent. This number of decimal digits is necessary to avoid misleading display of two floating-point numbers which only differ by one or a few least significant bits, but are represented identically.

C Library Proposal

Three new macros to be inserted just after **`FLT_DIG, DBL_DIG, LDBL_DIG`**

```
FLT_MAXDIG10 for float
DBL_MAXDIG10 for double
LDBL_MAXDIG10 for type long double
```

The number of base 10 digits required to ensure that values which differ by only one smallest (usually binary) unit in the last place (ulp) are always differentiated.

And as a non-normative note:

Values for these macros are usually conveniently derived from the number of significand (mantissa) binary digits, FLT_MANT_DIG, DBL_MANT_DIG or LDBL_MANT_DIG using the formula

```
max_decimal_digits = 2 + significand_digits * 3010/1000
```

For example:

```
#define FLT_MAXDIG10 (2+(FLT_MANT_DIG * 3010)/10000)
#define DBL_MAXDIG10 (2+ (DBL_MANT_DIG * 3010)/10000)
#define LDBL_MAXDIG10 (2+ (LDBL_MANT_DIG * 3010)/10000)
```

which yield the following values on typical implementations:

```
FLT_DIG 6, FLT_MAXDIG10 9
DBL_DIG 15, DBL_MAXDIG10 17
LDBL_DIG 19, LDBL_MAXDIG10 21
```

And a reference to Kahan's paper:

William Kahan http://http.cs.berkley.edu/~wkahan/ieee754status/ieee754.ps