IEEE 754 Support in C99

Jim Thomas
Hewlett-Packard Company
Cupertino, CA
PH:408 447 5781
FAX:408 447 4924
jwthomas@cup.hp.com
C99 Floating-Point History

IEEE 754 support a charter focus area for Numerical C Extensions Group (NCEG) - 1989

Participation and consultation from IEEE 754/854 members

NCEG Technical Report -- 1995

NCEG merged into C9x committee

C99 became standard in 1999
C99 Floating-Point Specification
Organization

Basic FP specification for all implementations, not just IEEE

- common API for FP and math library
- complex arithmetic and library

Annex F

- additional specification for IEC 60559 (IEEE 754) implementations
- IEEE 754 binding and compatible elementary functions
- conditionally normative

Annex G

- specification for complex arithmetic for IEC 60559 implementations
- informative
C99 Annex F
IEEE 754 Binding

Types

float $\rightarrow$ IEEE single
double $\rightarrow$ IEEE double
long double $\rightarrow$ IEEE double extended
else non-IEEE wide type, else IEEE double

Operators and functions

+, -, *, and / $\rightarrow$ IEEE add, subtract, multiply, and divide
sqrt() $\rightarrow$ square root
remainder() $\rightarrow$ remainder. remquo() same, with low order quotient bits.
rint() $\rightarrow$ rounds FP number to integer value (in the same precision)
nearbyint() $\rightarrow$ nearbyinteger in 854 appendix
conversions among floating types $\rightarrow$ IEEE conversions among FP precisions
conversions from integer to floating types $\rightarrow$ conversions from integer to FP
conversions from floating to integer types $\rightarrow$ IEEE-style conversions but
always round toward zero (inexact exception optional)
lrint() and llrint() $\rightarrow$ rounding mode conversions from floating point to
long int and long long int
Operators and functions (cont.)

translation time conversion of floating constants and `strtof()`, `strtod()`, `strtol()`, `fprintf()`, `fscanf()`, and related library functions

IEEE binary-decimal conversions. `strtol()` \rightarrow` conv function in 854 annex.

Correctly rounded binary decimal conversion is required, between widest supported IEEE type and decimal numbers with `DECIMAL_DIG` digits, where `DECIMAL_DIG` digits suffice to distinguish all binary FP values

relational and equality operators \rightarrow IEEE comparisons. Macro functions `isgreater()`, `isgreate Balequal()`, `isless()`, `islessequal()`, `islessgreater()`, `isunordered() \rightarrow “quiet” comparisons

`feclearexcept()`, `feraiseexcept()`, `fetestexcept()` \rightarrow test and alter IEEE exception status flags. `fegetexceptflag()` and `fesetexceptflag()` \rightarrow save and restore all status flags (including any auxiliary state).

Use with type `fexcept_t` and macros `FE_INEXACT`, `FE_DIVBYZERO`, `FE_UNDERFLOW`, `FE_OVERFLOW`, `FE_INVALID`

`fegetround()` and `fesetround()` \rightarrow select among IEEE rounding modes Macros `FE_TONEAREST`, `FE_UPWARD`, `FE_DOWNWARD`, `FE_TOWARDZERO \rightarrow IEEE rounding modes. Values 0, 1, 2, 3 of `FLT_ROUNDS` \rightarrow IEEE rounding modes
IEEE 754 Binding

Operators and functions (cont.)

fegetenv(), feholdexcept(), fesetenv(), and feupdateenv() → manage status flags and control modes, facilitate hiding exceptions
FENV_ACCESS ON pragma, with file or block effect, required for well defined behavior of code that reads flags or runs under non default modes
copysign() → copysign in 754 appendix
unary minus (-) → minus (-) operation in 754 appendix
scalbn() and scalbln() → scalb in 754 appendix (scalbln() has long int second parameter)
logb() → logb in 854 appendix. ilogb() like logb() except returns type int
nextafter() and nexttoward() → nextafter in appendix, except returns y if x = y. nexttoward() doesn’t clamp a wide direction argument
isnan() macro → isnan function in appendix. Inquiry macros are type generic
isfinite() macro → finite function in Appendix. Also there’s isinf() and isnormal()
signbit() and fpclassify() macros with number classification
   macros FP_NAN, FP_INFINITE, FP_NORMAL, FP_SUBNORMAL, FP_ZERO → class function in Appendix (except for signaling NaNs)
IEEE 754 Binding

Special values

INFINITY and NAN macros \(\rightarrow\) +infinity and a quiet NaN. Usable for static and aggregate initialization
sign respected for zero and infinity
I/O supports inf, infinity, nan, nan(n-char-sequence). Interpretation of n-char-sequence is implementation defined. Input case insensitive. User choice of upper or lower case (e.g. INF or inf) for output
nan() function takes “n-char-sequence” argument and constructs NaN at runtime

Not supported (though not disallowed)

trap handling (except with SIGFPE)
signaling NaNs
NaN significands (except for n-char-sequences)
compile-time mode and flag access

tried to specify trap handling and signaling NaNs, but found insufficient prior art and use, IEEE 754 guidance, inspiration
IEEE 754 Related Specification

Expression evaluation
Elementary functions
Complex arithmetic
IEEE 754 Related Features
Expression Evaluation

File or block pragma FP_CONTRACT allows or disallows contraction optimizations (e.g. fused multiply add synthesis). FP_CONTRACT ON can be default

Other value changing optimizations disallowed

fma() guarantees fused multiply add

3 well-defined expression evaluation methods

1) evaluate each operation and floating constant to semantic type
2) widen float operations and floating constants to double
3) widen float and double operations and floating constants to long double

Evaluation type may be wider than semantic type – wide evaluation does not widen semantic type

Assignments, casts, and argument passing convert to semantic type

FP_EVAL_METHOD macro identifies method in effect

Implementation may provide any, all, or none of these methods

“Widest-need” evaluation allowed but not specified. Specified in NCEG Technical Report but not in C99 because of lack of prior art
IEEE 754 Related Features
Wide Evaluation

Types float_t and double_t match evaluation types for float and double.

Inclusion of <tgmath.h> makes type of math function be determined by its argument.

Wide evaluation example:

```c
#include <tgmath.h>
float x, y, z;
float_t ss;
ss = x*x + y*y;
z = sqrt(ss);
```

computes $\sqrt{x^2+y^2}$ entirely in the evaluation type and converts to float only in the last assignment.

C99-portable code - uses wide evaluation if available.
IEEE 754 Related Features
Elementary Functions

Three fully supported real types: float double long double

C89 math library
acos asin atan atan2 cos sin tan cosh sinh tanh exp frexp ldexp log log10 modf pow sqrt ceil fabs floor fmod

C99 math additions
erf erfc lgamma tgamma hypot acosh asinh atanh cbrt expm1 ilogb log1p logb nextafter remainder rint isnan isinf signbit isnormal fpclassify isunordered isgreater isgreaterequal isless islessequal islessgreater copysign log2 exp2 fdim fmax fmin nan scalbn scalbln nearbyint round trunc remquo lrint lround llrint llround fma nexttoward

C99 floating-point environment library
feclearexcept fegetexceptflag feraiseexcept fesetexceptflag fetestexcept fegetround fesetround fegetenv feholdexcept fesetenv feupdateenv
IEEE 754 Related Features
Elementary Functions – Special Cases

(math_errhandling & MATH_ERREXCEPT) tests for 754-style exception flags
  - required by Annex F

(math_errhandling & MATH_ERRNO) tests for errno support

Annex F has detailed specification of special cases for real functions

IEEE 754 meaning of exceptions

exceptions required only under FENV_ACCESS ON

functions that are essentially always inexact are not required to raise inexact

functions may raise inexact if result is exact (implementation defined)

functions may raise underflow if result is tiny and exact (implementation defined)

functions may or may not honor rounding directions (implementation defined)

specifies numeric result instead of NaN if numeric result is useful in some significant

applications – CONTENTIOUS
IEEE 754 Related Features
Elementary Functions – Special Cases

Contentious special cases:

atan2(±0, -0) = ±pi, atan2(±0, +0) = ±0
atan2(±inf, -inf) = ±3pi/4, atan2(±inf, +inf) returns ±pi/4
hypot(±inf, y) = +inf, even if y is a NaN
pow(-1, ±inf) = 1
pow(+1, x) = 1 for any x, even a NaN
pow(x, ±0) = 1 for any x, even a NaN
pow(-inf, y) = +0 for y<0 and not an odd integer
pow(-inf, y) = -inf for y an odd integer > 0
pow(-inf, y) = +inf for y>0 and not an odd integer

if just one argument is a NaN, the fmax and fmin functions return the other argument
IEEE 754 Related Features

Complex Arithmetic

Three complex types: float complex, double complex, long double complex

Annex G specifies three imaginary types: float, double, and long double imaginary

Operands not promoted to a common type domain (real, imaginary, complex)

e.g. \( r(u + vi) = ru + rvi \), not \( (r + 0i)(u + vi) \)

provides natural efficiency and better treatment of special values

e.g. \( i \cdot i = -1 \), not \( (+0i)(0 + i)(+0i)(0 + i) = \text{NaN} + \text{NaN}i \)

Infinity properties

for \( z \) nonzero and finite

\[
\begin{align*}
*z & = *z & /z & = /0 = 0/z = \text{NaN} \\
z/ & = 0/ = 0 & z/0 = & |z| = \text{NaN}
\end{align*}
\]
even for complex and imaginary \( z \), 0s, and infinities

a complex value with at least one infinite part is regarded as infinite (even if the other part is NaN)

CX_LIMITED_RANGE ON (file or block) pragma allows implementation to

deploy simpler code and forgo infinity properties
IEEE 754 Related Features
Complex Arithmetic, Functions

Sample implementations of multiply and divide use just one isnan test to condition special case code

Multiply and divide must raise deserved exceptions and may raise spurious ones

Imaginary unit I
- float imaginary constant
- $x + y*I$, where $x$, $y$ are of same real type, requires no actual FP ops

Complex library
- cacos, casin, catan, ccos, csin, ctan, cacosh, casinh, catanh, ccosh, csinh, ctanh
- cexp, clog, csqrt, cabs, cpow, carg, conj, cimag, cproj, creal

Inclusion of <tgmath.h> makes math functions generic for real, complex, and imaginary.

- $\exp(z) = \text{cexp}(z)$, if $z$ is float complex
- $\sin(y*I) = \sinh(y)*I$, if $y$ is double
- $\cos(y*I) = \cosh(y)$, if $y$ is long double
Annex G specifies non NaN results for special cases where useful for preserving magnitude or direction information – CONTENTIOUS

cexp(-inf+iNaN) = ±0±i0 (where the signs of the real and imaginary parts of the result are unspecified)

csqrt(x+iinf) returns +inf+iinf, for all x (including NaN)

cacos(+inf+iinf) returns pi/4-iinf

creal(x, iNaN) = x
C99 Support for IEEE 754 Reception

C99 represents a 10 year, good faith effort by a language standard group, with lots of help from the numerical community, to support IEEE 754

Being picked up by next Unix standard

Impact on next C++ standard TBD

Several vendors have implemented, or are implementing, all or part

HP-UX C for Itanium has essentially all of C99 FP

Careful, useful (reasonable performance) implementation requires great attention to detail, beyond what can be expected of compiler teams

ROI seen as greater for performance work

Customer demand (for features beyond basics) seen as low, customer appreciation TBD

Needs support from numerical community

affirmation of value

demonstration of utility