ISO/IEC TS 18661 OVERVIEW

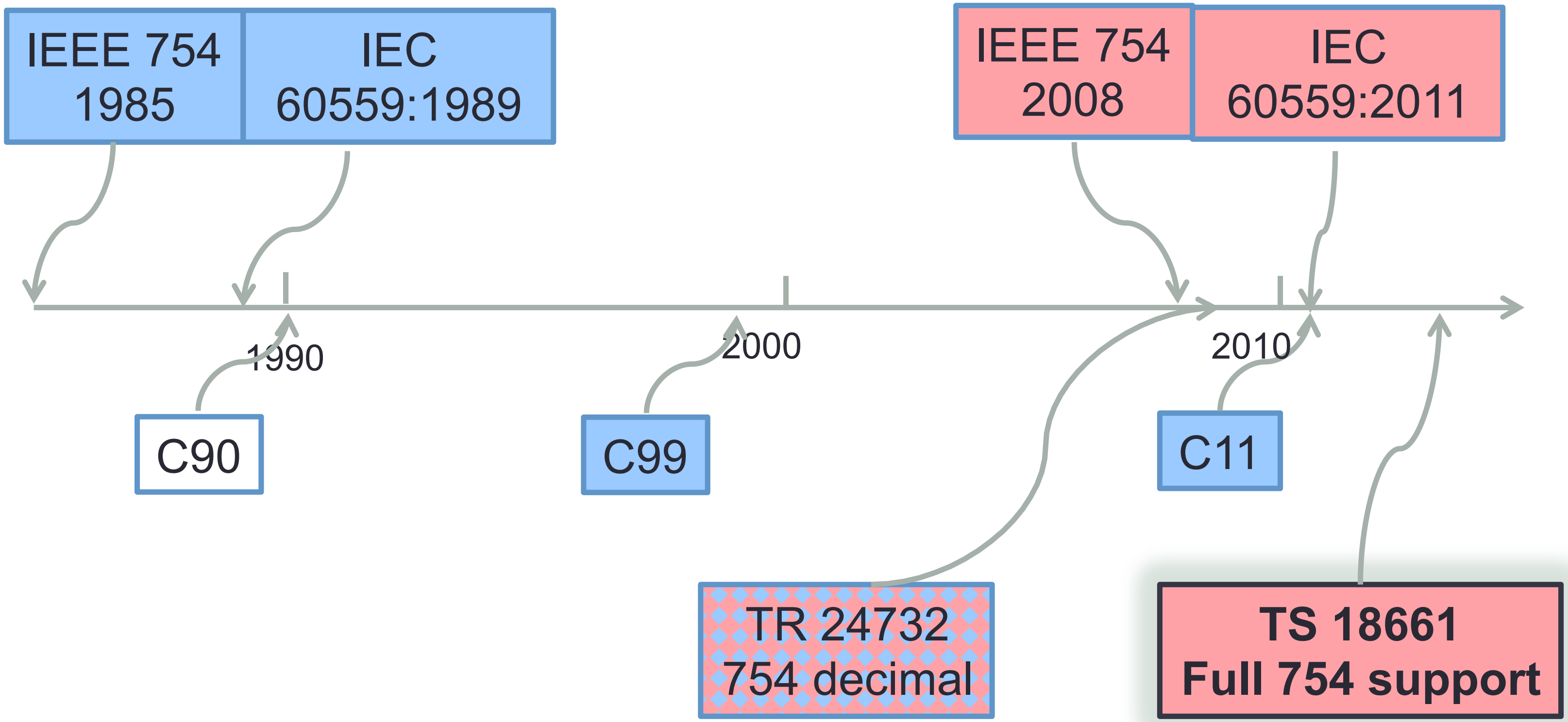
23rd IEEE Symposium on Computer Arithmetic – ARITH23
July 13, 2016

Jim Thomas
jaswthomas@sbcglobal.net
Davis, CA USA

ISO/IEC Technical Specification 18661

C extensions to support IEEE 754-2008

Floating-point and C standards



IEEE 754 1985

IEC 60559:1989

IEEE 754 2008

IEC 60559:2011

1990

2000

2010

C90

C99

C11

TR 24732 754 decimal

TS 18661 Full 754 support

Background

Specify a C binding for IEEE 754-2008

- Work began 2009
- Under direction of ISO/IEC JTC1/SC22/WG14 – C
- Expertise in floating-point and language standards, compilers, libraries

Principles

- Support all of IEEE 754-2008, as-is
- Specify as changes to C11
- Use existing C mechanisms, minimize language invention
- Develop specification in parts, to pipeline process
- Supersede TR 24732
- Deliver an ISO/IEC Technical Specification

Status

- In five parts
  1. Binary floating-point arithmetic
  2. Decimal floating-point arithmetic
  3. Interchange and extended types
  4. Supplementary functions
  5. Supplementary attributes
- Parts 1-4 published in 2014-2015
- Part 5 approved, publication expected in 2016

Publications

- ISO/IEC TS 18661-1:2014, Information technology — Programming languages, their environments and system software interfaces — Floating-point extensions for C — Part 1: Binary floating-point arithmetic

- ISO/IEC TS 18661-2:2015, Information technology — Programming languages, their environments and system software interfaces — Floating-point extensions for C — Part 2: Decimal floating-point arithmetic

- ISO/IEC TS 18661-3:2015, Information technology — Programming languages, their environments and system software interfaces — Floating-point extensions for C — Part 3: Interchange and extended types

- ISO/IEC TS 18661-4:2015, Information Technology — Programming languages, their environments, and system software interfaces — Floating-point extensions for C — Part 4: Supplementary functions

Expected

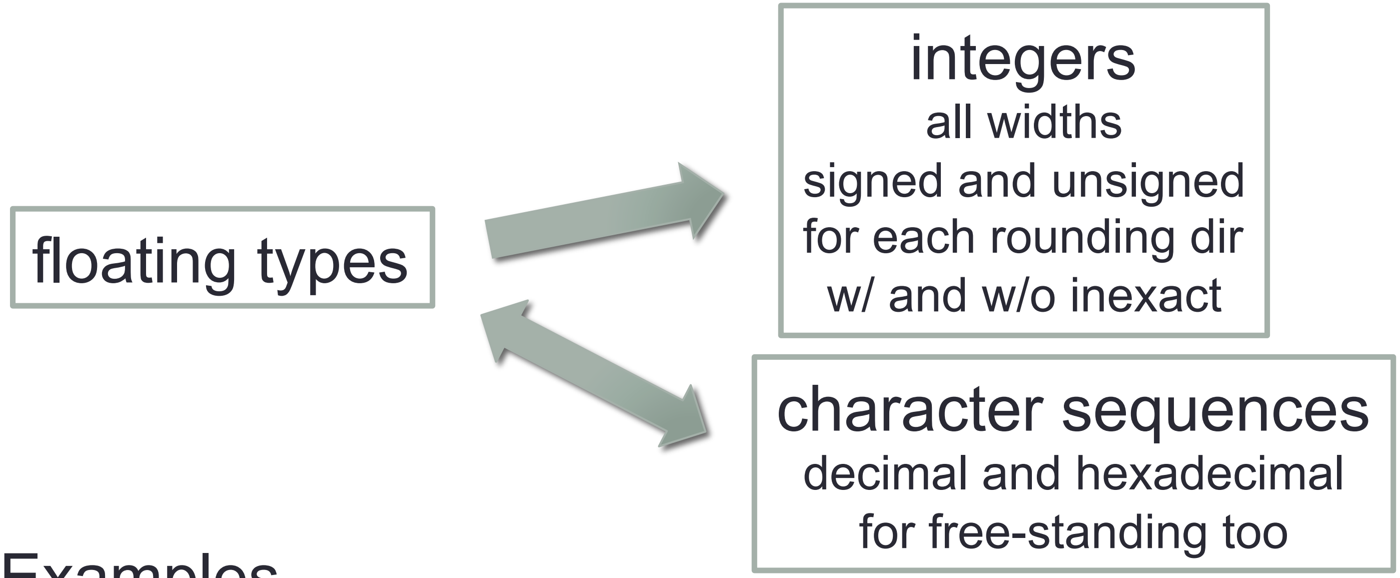
- ISO/IEC TS 18661-5:2016, Information Technology — Programming languages, their environments, and system software interfaces — Floating-point extensions for C — Part 5: Supplementary attributes

Part 1

- TS 18661-1 – Binary floating-point arithmetic
- Required parts of IEEE 754-2008 for binary formats
- Binds 754 binary32 and binary64 formats to C `float` and `double` types
- Binds all 754 required operations to C operators and library functions
- Some example of new features …

Part 1

Conversions



floating types

integers
all widths
signed and unsigned
for each rounding dir
w/ and w/o inexact

character sequences
decimal and hexadecimal
for free-standing too

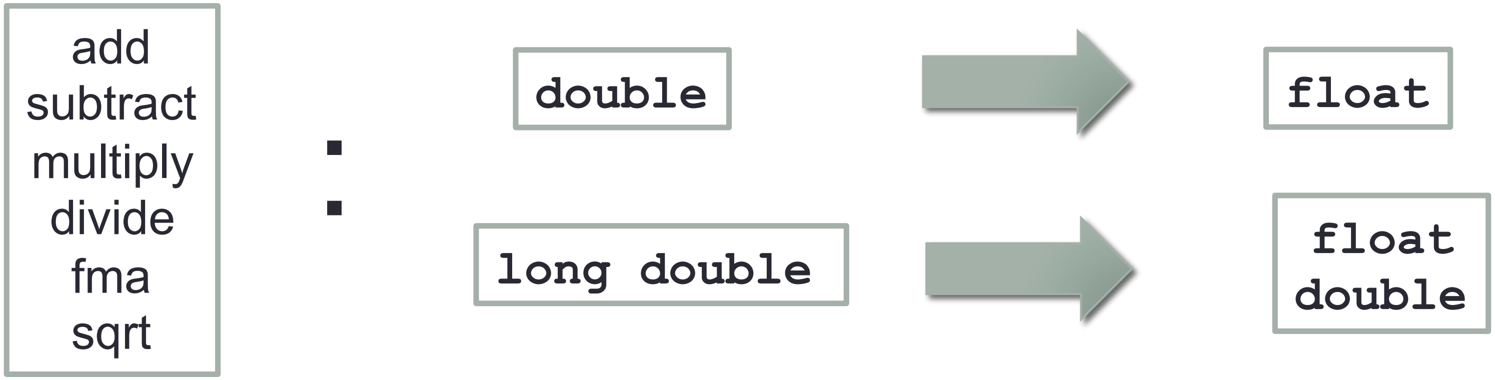
Examples

```
intmax_t fromfp(double x, int round, unsigned int width);
```

```
int strfromd(char * restrict s, size_t n,
             const char * restrict format, double fp);
```

Part 1

Functions that round results to narrower type



add
subtract
multiply
divide
fma
sqrt

:

`double` → `float`

`long double` → `float` `double`

Example

```
float ffma(double x, double y, double z);
```

Part 1

More classification and comparison macros

`issubnormal()`
`issignaling()`
`iscanonical()`
`iseqsig()` – test equality, signal invalid on NaN input

Better NaN support

`getpayload()`   **`setpayload()`**   **`setpayloadsig()`**
Signaling NaN macros
Optional signaling NaN support

More facilities for exception flags and modes

`fesetexcept()`   **`fetestexceptflag()`**
`femode_t`   **`fegetmode()`**   **`fesetmode()`**

Part 1

Other functions, including

`roundeven()` – 754 round to nearest (ties to even) integer in floating format
`nextup()` – next larger representable number
`nextdown()` – next smaller representable number
`fmaxmag()` – argument of maximum magnitude
`fminmag()` – argument of minimum magnitude
`totalorder()` – total ordering of canonical encodings
`totalordermag()` – total ordering of magnitudes of canonical encodings

Part 1

Binds 754 rounding direction attribute to new constant mode pragma

```
{
  #pragma STDC FENV_ROUND FE_TOWARDZERO
  z = sqrt(x + y);
}
```

An alternative to dynamic rounding mode

```
{
  int save_round;
  save_round = fegetround();
  fesetround(FE_TOWARDZERO);
  z = sqrt(x + y);
  fesetround(save_round);
}
```

Part 2

- TS 18661-2 – Decimal floating-point arithmetic
- Required parts of IEEE 754 for decimal
- Full C and 754 support for 32, 64, 128 bit decimal formats
  - Types
  - Built-in operator
  - Functions, macros, pragmas
  - Constants
  - I/O width modifiers
- Including support for 754 quantum for decimal
  - Exact operators and math functions produce the preferred quantum exponent, e.g., 1.07 + 0.13 = 1.20, not 1.2
  - `%a`, `%A` output and all input preserve quantum exponents

Part 2

```
...
_Decimal64 rate = 175.DD, hours, fee, total = 0.00DD;
...
   scanf(“%De”, &hours);
   {
     #pragma STDC FENV_DEC_ROUND FE_DEC_TONEARESTFROMZERO
     fee = rate * hours;
     fee = quantized64(fee, 0.00DD); // round to cents
   }
   total += fee;
...
printf(“%Da\n”, total);
...
```

Part 2

Uses encode/decode functions and **`unsigned char`** arrays to handle external data in either of the two 754 encodings of decimal data

```
_Decimal32 x, y;
unsigned char encoding[32/8];
```

… read *decimal-encoded decimal* into **`encoding`**

```
decodedecd32(&x, encoding);
```

... use **`x`**, compute **`y`**

```
encodebind32(encoding, &y);
```

… write *binary-encoded decimal* from **`encoding`**

Conformance

- Implementation may conform to Part 1 or Part 2 or both
- Then may conform to Parts 3, 4, and 5 in any combination
- Supportable by hosted or free-standing C implementations

Part 3

- TS 18661-3 – Interchange and extended types
- Optional IEEE 754 interchange and extended formats
- Interchange formats may be arithmetic or not
- Full* 754 and C support for unlimited number of fixed width arithmetic interchange formats, including float16
- And for extended formats which have more range and precision than basic formats in Parts 1 and 2
- Mechanisms for interchange of data in 754 formats that are supported but not as arithmetic
- Binary and decimal formats

* I/O with strings using `strto` and `strfrom` functions, instead of with more width modifiers

Part 3

C *real floating types*

standard floating types
`float`
`double`
`long double`

Other floating types	***binary***	***decimal***
interchange	`_Float`*N*, *N*=16,32,64,128, 160,...	`_Decimal`*N*, *N*=32,64,96,128, 160,…
extended	`_Float`*N*`x`, *N*=32,64,128	`_Decimal`*N*`x`, *N*=64,128

Part 3

- Non-arithmetic interchange formats supported by conversion functions and `unsigned char` arrays
- Example – suppose implementation supports float16 as non-arithmetic format …

```
_Float32 x;
unsigned char enc16[16/8];
unsigned char enc32[32/8];

… store float16 encoding in enc16

f32encf16(enc32, enc16);
decodef32(&x, enc32);


...
```

Part 4

- TS 18661-4 – Supplementary functions
- Mathematical functions
  - 754 recommends correct rounding
  - TS adds all the ones not already in C11
  - TS reserves names for correctly rounded versions

Defines	**`double sinpi(double x);`**
Reserves	**`crsinpi`**

- Reduction operations
  - sum reductions
  - scaled products
  - 754 does not prescribe correct rounding, or reproducibility

Part 4

New math functions

`exp2m1`	`rsqrt`	`asinpi`
`exp10`	`compound`	`atanpi`
`exp10m1`	`rootn`	`atan2pi`
`logp1`	`pown`	`cospi`
`log2p1`	`powr`	`sinpi`
`log10p1`	`acospi`	`tanpi`

Part 4

New reduction functions

reduc_sum	scaled_prod
reduc_sumabs	scaled_prodsum
reduc_sumsq	scaled_proddiff
reduc_sumprod	

Examples

```
double reduc_sum(size_t n, const double p[static n]);
```

returns $\Sigma_{i=0,\mathbf{n}-1}\mathbf{p[i]}$

```
double scaled_prodsum(size_t n, const double p[static restrict n],
            const double q[static restrict n], intmax_t * restrict sfptr);
```

returns pr such that $pr \times b^{sf} = \Pi_{i=0,\mathbf{n}-1}(\mathbf{p[i]} + \mathbf{q[i]})$

Part 5

- TS 18661-5 – Supplementary attributes
- 754-recommended attributes
- Way for user to specify alternate semantics for a block of code
  - Evaluation formats (wide evaluation)
  - Optimization controls
  - Reproducible results
  - Alternate exception handling
  - (Required attributes for constant rounding modes in Parts 1 and 2)
- All done with pragmas, like other FP attributes already in C

Part 5

Evaluation formats

```
#pragma STDC FENV_FLT_EVAL_METHOD width
#pragma STDC FENV_DEC_EVAL_METHOD width
```

width matches a value of the **`FLT_EVAL_METHOD`** or **`DEC_EVAL_METHOD`** macro

```
{
   #pragma STDC FENV_FLT_EVAL_METHOD 0
   … operations evaluated to type (no extra range or precision)
}
```

Part 5

Optimization controls

`#pragma STDC FENV_ALLOW_VALUE_CHANGING_OPTIMIZATION` *on-off-switch*

`#pragma STDC FENV_ALLOW_ASSOCIATIVE_LAW` *on-off-switch*

`#pragma STDC FENV_ALLOW_DISTRIBUTIVE_LAW` *on-off-switch*

`#pragma STDC FENV_ALLOW_MULTIPLY_BY_RECIPROCAL` *on-off-switch*

`#pragma STDC FENV_ALLOW_ZERO_SUBNORMAL` *on-off-switch*

`#pragma STDC FENV_ALLOW_CONTRACT_FMA` *on-off-switch*

`#pragma STDC FENV_ALLOW_CONTRACT_OPERATION_CONVERSION` *on-off-switch*

`#pragma STDC FENV_ALLOW_CONTRACT` *on-off-switch*

on-off-switch is one of **`ON`**, **`OFF`**, **`DEFAULT`**

Part 5

Reproducible results

```
#pragma STDC FENV_REPRODUCIBLE on-off-switch
```

implies the effects of

```
#pragma STDC FENV_ACCESS ON
#pragma STDC FENV_ALLOW_VALUE_CHANGING_OPTIMIZATION OFF
#pragma STDC FENV_FLT_EVAL_METHOD 0
#pragma STDC FENV_DEC_EVAL_METHOD 1
```

TS provides guidance for the programmer and recommends compiler diagnostics

Part 5

Alternate exception handling

- Deal with exceptions directly, rather than through flags

`#pragma STDC FENV_EXCEPT` *action except-list*

action is one of

`DEFAULT` `NO_FLAG` `OPTIONAL_FLAG` `ABRUPT_UNDERFLOW`

and these that change control flow ASAP

`BREAK`

`TRY` `CATCH`

and these that change control flow and are deterministic

`DELAYED_TRY` `DELAYED_CATCH`

Part 5

```
...
#pragma STDC FENV_EXCEPT TRY FE_DIVBYZERO, FE_OVERFLOW
{
    for (int i=0; i<LEN; i++) {
        f[i] = 1.0 / d[i];
    }
}
#pragma STDC FENV_EXCEPT CATCH FE_DIVBYZERO
{
    printf("divide-by-zero\n");
}
#pragma STDC FENV_EXCEPT CATCH FE_OVERFLOW
{
    printf("overflow\n");
}
...
```

ISO/IEC TS 18661

- C extensions to support IEEE 754-2008
- Fifth and final part publishes this year
- Substantial portions have been and are being implemented
- Included in Cyy? Which parts?
- Good for IEEE 754-2018?