Proposed update for TS 18661-4
WG14 N2274

Title:	Augmented arithmetic functions
Author:	C FP Group
Date:	2018-06-30
Proposal category:	New feature
Target audience:	IEEE 754-201x, extra precision, reproducible summation

The 2018 update to IEEE 754 adds optional operations for augmented arithmetic. This is a proposal to update TS 18661-4 to specify a C binding for these operations.

Changes to TS 18661:

After clause 8, insert the clause:

8a Functions for augmented arithmetic in `<math.h>`

This clause specifies changes to C11 + TS18661-1 + TS18661-2 + TS18661-3 to include functions that support operations for augmented arithmetic, as recommended by IEC 60559.

Changes to C11 + TS18661-1 + TS18661-2 + TS18661-3:

After F.10.12, add:

F.10.13 Augmented arithmetic

[1] This subclause specifies types and functions for **`<math.h>`** for augmented arithmetic, as recommended by IEC 60559 for its binary formats. These functions are not specified for decimal types.

[2] The functions in this subclause round to nearest with ties toward zero, a rounding direction specified by IEC 60559 for use by augmented arithmetic operations. Thus, results are independent of dynamic and constant rounding direction modes.

[3] The types are structures for returning two floating-point values:

```
struct daug_t { double h; double t; };
struct faug_t { float h; float t; };
struct ldaug_t { long double h; long double t; };
struct _fNaug_t { _FloatN h; _FloatN t; };
struct _fNxaug_t { _FloatNx h; _FloatNx t; };
```

The *corresponding real type* of the structure refers to the type of the members.

F.10.13.1 The `augadd` functions

Synopsis

[1]
```
#define __STDC_WANT_IEC_60559_FUNCS_EXT__
#include <math.h>
struct daug_t augadd(double x, double y);
struct faug_t augaddf(float x, float y);
struct ldaug_t augaddl(long double x, long double y);
struct _fNaug_t augaddfN(_FloatN x, _FloatN y);
struct _fNxaug_t augaddfNx(_FloatNx x, _FloatNx y);
```

Description

[2] The **`augadd`** functions compute two result values:

`h`: the sum **`x`** + **`y`** rounded to the type using round-to-nearest with ties toward zero;

`t`: the error in **`h`** as a computation of **`x`** + **`y`**.

If **`h`** is a non-zero finite number, **`t`** has the value **`x`** + **`y`** – **`h`** (which is exactly representable in the type). If **`h`** is zero, **`t`** has the value of **`h`** (hence both have the same sign). If **`h`** is infinite, **`t`** has the value of **`h`**. If **`h`** is a NaN, **`t`** is the same NaN.

[3] These functions raise floating-point exceptions like the computation of **`h`**, except that they raise the "inexact" floating-point exception only when the computation of **`h`** overflows.

[4] A range error occurs when the computation of **`h`** overflows. The "invalid" floating-point exception is raised and a domain error occurs when the arguments are infinities with different signs.

Returns

[5] These functions return the sum and error in a structure.

F.10.13.2 The `augsub` functions

Synopsis

[1]
```
#define __STDC_WANT_IEC_60559_FUNCS_EXT__
#include <math.h>
struct daug_t augsub(double x, double y);
struct faug_t augsubf(float x, float y);
struct ldaug_t augsubl(long double x, long double y);
struct _fNaug_t augsubfN(_FloatN x, _FloatN y);
struct _fNxaug_t augsubfNx(_FloatNx x, _FloatNx y);
```

Description

[2] The **augsub** functions compute two result values:

h: the difference **x** − **y** rounded to the type using round-to-nearest with ties toward zero;

t : the error in **h** as a computation of **x** − **y**.

If **h** is a non-zero finite number, **t** has the value **x** – **y** – **h** (which is exactly representable in the type). If **h** is zero, **t** has the value of **h** (hence both have the same sign). If **h** is infinite, **t** has the value of **h**. If **h** is a NaN, **t** is the same NaN.

[3] These functions raise floating-point exceptions like the computation of **h**, except that they raise the "inexact" floating-point exception only when the computation of **h** overflows.

[4] A range error occurs when the computation of **h** overflows. The "invalid" floating-point exception is raised and a domain error occurs when the arguments are infinities with the same sign.

Returns

[5] These functions return the difference and error in a structure.

F.10.13.3 The augmul functions

Synopsis

[1]
```
#define __STDC_WANT_IEC_60559_FUNCS_EXT__
#include <math.h>
struct daug_t augmul(double x, double y);
struct faug_t augmulf(float x, float y);
struct ldaug_t augmull(long double x, long double y);
struct _fNaug_t augmulfN(_FloatN x, _FloatN y);
struct _fNxaug_t augmulfNx(_FloatNx x, _FloatNx y);
```

Description

[2] The **augmul** functions compute two result values:

h: the product **x** × **y** rounded to the type using round-to-nearest with ties toward zero;

t : the error in **h** as a computation of **x** × **y**.

If **h** is a nonzero finite number, **t** is **x** × **y** – **h** rounded to the type using round-to-nearest with ties toward zero. (The computation of **t** will be exact unless the magnitude of **x** × **y** – **h** is too small.) If **h** is zero, **t** has the value of **h** (hence both have the same sign). If **h** is infinite, **t** has the value of **h**. If **h** is a NaN, **t** is the same NaN.

[3] These functions raise floating-point exceptions like the computation of **h**, with the following additional specification. They raise the "underflow" floating-point exception when and only when the computation of **t** underflows. They raise the "inexact" floating-point exception when and only when the computation of **h** overflows or the computation of **t** is inexact.

[4] A range error occurs when the computation of **h** overflows and may occur when the computation of **t** underflows. A domain error occurs when the computation of **h** is invalid.

Returns

[5] These functions return the product and error in a structure.

Straightforward updates to add the new functions above to the lists in 5.3 and the table in clause 6, in TS 18661-4, are needed.

The following changes allow type-generic math to apply to the functions in this clause.

In 7.25#5, to the list of macro names, add macro names for the functions for augmented arithmetic: **augadd**, **augsub**, **augmul**.

In 7.25#5, change:

> If all arguments for generic parameters are real, then use of the macro invokes a real function; otherwise, use of the macro results in undefined behavior.

to:

> If all arguments for generic parameters are real, then use of the macro invokes a function returning a real type or a function returning a structure whose members are of one real type (the corresponding real type); otherwise, use of the macro results in undefined behavior.

In 7.25#7, add to the list of examples:

augadd(d, ld)	**augaddl(d, ld)**