Farance Inc.

doc: **C Binding of ISO 10967 (LIA-1)**
WG14/N528 X3J11/95-129
file: **`language-independent/lia-1.*`**

date: **1995-12-22**

from: **Frank Farance**
+1 212 486 4700
frank@farance.com

William Rugolsky, Jr.
+1 212 486 4700
rugolsky@farance.com

ABSTRACT

This is the second preliminary proposal for the inclusion of a C binding of language independent arithmetic (LIA-1) as defined in ISO 10967-1:1994. LIA-1 specifies a parameterized model of arithmetic computation. The purpose of LIA-1 is to provide a known environment in conforming implementations across platforms and languages for applications requiring numeric computation. Overall, the C binding of LIA-1 doesn't affect existing programs but new programs will achieve a higher degree of portability on LIA-1 systems. The impact of the changes are: adding some macros, adding a handful of library functions, and requiring the implementation to document certain features of its arithmetic. This revision of the proposal includes only the LIA-1 C binding without the rationale. It is expected that this proposal will be revised and harmonized with the floating point extensions proposal.

CONTENTS

1. IMPACT TO STANDARD C

This section provides an rough draft of the wording that would be added to the Standard to support LIA-1. The definitions of the parameters (e.g., `INT_MODULO`) have been omitted for the sake of clarity at this phase of review: most of the definitions are obvious or defined in LIA-1. Of course, the definitions would be included in future proposals and in the final Standards wording.

The following wording would be added to the Standard:

1.1 Language Independent Arithmetic <lia.h>

An implementation shall conform to all the requirements of LIA-1 (ISO 10967-1:1994) unless otherwise specified in this clause.

NOTE: The operations or parameters marked † are not part of Standard C and are enhancements required by LIA-1.

1.1.1 Boolean Type

The LIA-1 data type Boolean is implemented in the C data type `int` (1 == true and 0 == false).

1.1.2 Integral Types

The integral types `int`, `long`, `unsigned int`, and `unsigned long` conform to LIA-1.

NOTE: The conformity of `short` and `char` (signed or unsigned) is not relevant since values of these types are promoted to `int` (signed or unsigned) before computations are done.

1.1.2.1 LIA-1 Parameters

The parameters for the LIA-1 integer data types can be accessed by the following:

maxint	`INT_MAX`, `LONG_MAX`, `UINT_MAX`, `ULONG_MAX`.
minint	`INT_MIN`, `LONG_MIN`.
modulo	`INT_MODULO`†, `LONG_MODULO`†.

The parameter *bounded* is always true, and is not provided. The parameter *minint* is always 0 for the unsigned types, and is not provided for those types. The parameter *modulo* is always true for the unsigned types, and is not provided for those types.

1.1.2.2 LIA-1 Operations

The integer operations are the following:

addI	`x + y`.
subI	`x - y`.
mulI	`x * y`.

divI	`x / y`.
remI	`x % y`.
modaI	`modulo(x,y)`†, `lmodulo(x,y)`†.
modpI	No binding.
negI	`- x`.
absI	`abs(x)`, `labs(x)`.
signI	`sgn(x)`†, `lsgn(x)`†.
eqI	`x == y`.
neqI	`x != y`.
lssI	`x < y`.
leqI	`x <= y`.
gtrI	`x > y`.
geqI	`x >= y`.

where `x` and `y` are expressions of the same integral type.

The C Standard permits *divI* and *remI* (`/` and `%`) to be implemented using either round toward minus infinity (*divfI*) or toward zero (*divtI/remtI*). The implementation shall choose the same rounding for both and document the choice.

In C9X, the signed integer division proposal will require all signed division to round toward zero.

1.1.3 Floating Types

The floating types `float`, `double`, and `long double` conform to LIA-1.

1.1.3.1 LIA-1 Parameters

The parameters for a floating point data type can be accessed by the following:

r	`FLT_RADIX`.
p	`FLT_MANT_DIG`, `DBL_MANT_DIG`, `LDBL_MANT_DIG`.
emax	`FLT_MAX_EXP`, `DBL_MAX_EXP`, `LDGL_MAX_EXP`.
emin	`FLT_MIN_EXP`, `DBL_MIN_EXP`, `LDBL_MIN_EXP`.
denorm	`FLT_DENORM`†, `DBL_DENORM`†, `LDBL_DENORM`†.
iec_559	`FLT_IEC_559`†, `DBL_IEC_559`†, `LDBL_IEC_559`†.

The `*_DENORM` macros and `*_IEC_559` macros represent booleans and have values 1 or 0.

The derived constants for the floating types are accessed by the following:

fmax	`FLT_MAX, DBL_MAX, LDBL_MAX.`
fminN	`FLT_MIN, DBL_MIN, LDBL_MIN.`
fmin	`FLT_TRUE_MIN†, DBL_TRUE_MIN†, LDBL_TRUE_MIN†.`
epsilon	`FLT_EPSILON†, DBL_EPSILON†, LDBL_EPSILON†.`
rnd_error	`FLT_RND_ERR†, DBL_RND_ERR†, LDGL_RND_ERR†.`
rnd_style	`FLT_ROUNDS.`

1.1.3.2 LIA-1 Rounding Styles

The C Standard requires all floating types use the same radix and rounding style, so that only one identifier for each is provided in the LIA-1 binding.

The `FLT_ROUNDS` parameter corresponds to the LIA-1 rounding styles:

truncate	`FLT_ROUNDS == 0.`
nearest	`FLT_ROUNDS == 1.`
other	`FLT_ROUNDS != 0 && FLT_ROUNDS != 1.`

NOTE: — The definition of `FLT_ROUNDS` has been extended to cover the rounding style used in all LIA-1 operations, not just addition.

1.1.3.3 LIA-1 Operations

The floating point operations are:

addF	`x + y.`
subF	`x - y.`
mulF	`x * y.`
divF	`x / y.`
negF	`- x.`
absF	`fabsf(x)†, fabs(x), fabsl(x)†.`
signF	`fsgnf(x)†, fsgn(x)†, fsgnl(x)†.`
exponentF	`exponf(x)†, expon(x)†, exponl(x)†.`
fractionF	`fractf(x)†, fract(x)†, fractl(x)†.`
scaleF	`scalef(x,n)†, scale(x,n)†, scalel(x,n)†.`
succF	`succf(x)†, succ(x)†, succl(x)†.`
predF	`predf(x)†, pred(x)†, precl(x)†.`

ulpF `ulpf(x)†, ulp(x)†, ulpl(x)†.`

truncF `truncto f(x,n)†, truncto(x,n)†, truncto l(x,n)†.`

roundF `roundtof(x,n)†, roundto(x,n)†, roundtol(x,n)†.`

intpartF `intprtf(x)†, intprt(x)†, intprtl(x)†.`

fractpartF `frcprtf(x)†, frcprt(x)†, frcprtl(x)†.`

eqF `x == y.`

neqF `x != y.`

lssF `x < y.`

leqF `x <= y.`

gtrF `x > y.`

geqF `x >= y.`

where `x` and `y` are expressions of the same floating point type, and `n` is of type `int`.

NOTE: *scaleF* can be computed using the `ldexp` library function, only if `FLT_RADIX==2`.

NOTE: The Standard C function `frexp` differs from *exponentF* in that no notification is raised when the argument is 0.

1.1.3.4 LIA-1 Indicators

The following indicators shall be provided a one method of notification (see LIA-1 subclause 6.1.2).

integer_overflow `INT_OVERFLOW†.`

floating_overflow `FLT_OVERFLOW†.`

underflow `UNDERFLOW†.`

undefined `UNDEFINED†.`

The values representing individual indicators shall be distinct non-negative powers of two. The empty set is denoted by `0`. Other indicator subsets are named by combining individual indicators using bit-or. For example, the LIA-1 indicator subset

```
{floating_overflow, underflow, integer_overflow}
```

would be denoted by the expression

```
FLT_OVERFLOW | UNDERFLOW | INT_OVERFLOW
```

The indicator interrogation and manipulation operations are:

set_indicators `set_indicators(i)†.`

clear_indicators `clear_indicators(i)`†.

test_indicators `test_indicators(i)`†.

current_indicators `current_indicators()`†.

where `i` is an expression of type `unsigned int` representing an LIA-1 indicator subset.

The implementation shall provide an alternative of notification through termination with a ''hard-to-ignore'' message (see LIA-1 subclause 6.1.3).

1.1.4 Type Conversions

LIA-1 operations shall be provided in all floating types.

The LIA-1 type conversions are the following type casts:

cvtF→I `(int) x, (long) x, (unsigned int) x, (unsigned long) x.`

cvtI'→I `(int) x, (long) x, (unsigned int) x, (unsigned long) x.`

cvtI→F `(float) x, (double) x, (long double) x.`

cvtF'→F `(float) x, (double) x, (long double) x.`

2. OPEN ISSUES

1. The C Standard requires that float to integer conversions round toward zero. An implementation that wishes to conform to LIA-1 must use round to nearest for conversions to a floating point type.
2. Harmonize this proposal with current floating point and complex proposals.
3. Provide detailed definition of new macros and functions required by LIA-1.
4. Add standards wording for each of the functions and macros.