TS 18661 Part 5 Supplementary attributes

WG 14 N1975

2015-10-20

IEC 60559 attributes

- N1974: draft TS 18661-5 – Supplementary attributes
- Updates N1919 presented in April
- Now functionally complete
- Format and boilerplate text for ISO

IEC 60559 attributes - review

- Constant modes for floating-point semantics
- Program specifies modes to apply to blocks
- IEC 60559 requires attributes for
  Rounding direction
- Recommends attributes for
  Evaluation formats
  Optimization control
  Reproducible code
  Alternate exception handling

C support for attributes - review

- Floating-point pragmas in <fenv.h>
- Rounding direction pragmas in parts 1 and 2
- Pragmas for recommended attributes in part 5
- All similar in form and scope to STDC pragmas in C standard

Evaluation formats - review

- #pragma STDC FENV_FLT_EVAL_METHOD *width*
  for standard and binary types
- *width* reflects a possible value of FLT_EVAL_METHOD macro
- Required support for *width* values -1, 0, and DEFAULT
- Other *width* values optional
- Similar FENV_DEC_EVAL_METHOD for decimal types
- Required support for decimal *width* values -1, 1, and DEFAULT

Evaluation formats – key changes

- P 3, 4: Interaction between evaluation method macros and pragmas
  - Macro values reflect the evaluation method in use, which be might set by a pragma
  - Macro shall not be used in #if and #elif expressions where a pragma is in effect
- P 5: _t types have default evaluation formats, but have corresponding type-like macros that (unless undefined) expand to types with the evaluation formats where used

Evaluation formats – key changes (2)

P 5: New macro user can define before including <tgmath.h> to make tgmath macros behave like built-in operators with respect to evaluation formats: __STDC_TGMATH_OPERATOR_EVALUATION__

- <tgmath.h> macros do not narrow arguments and they return results in evaluation formats
- Does not affect semantic types (just like operators)

Optimization control - review

- Allow/disallow value-changing optimizations (transformations)
- #pragma STDC FENV_ALLOW_... *on-off-switch*
- VALUE_CHANGING_OPTIMIZATION allows all the following, which can also be allowed separately
- ASSOCIATIVE_LAW
- DISTRIBUTIVE_LAW
- MULTIPLY_BY_RECIPROCAL

  A / B = A x (1/B)

Optimization control (2) - review

- ZERO_SUBNORMAL

  allow replacing subnormal operands and results with 0
- CONTRACT_FMA

  contract (compute with just one rounding) A x B + C
- CONTRACT_OPERATION_CONVERSION

  e.g., F = D1 * D2 and F = sqrt(D)
- CONTRACT

  all contractions

  equivalent to FP_CONTRACT pragma in <math.h>

Optimization control – key changes

P 8: Clarification about identities allowed for optimization

- Identities that are valid for IEC 60559 arithmetic, e.g.,

  $x + y = y + x$

- Identities derived from allowed identities, e.g., allowed associative law also allows

  $x + (y - z) = (x + y) - z$

  $x + (z + y) = (x + y) + z$

- Allowed distributive law explicitly includes:

  $x \times (y + z) = (x \times y) + (x \times z)$

  $x \times (y - z) = (x \times y) - (x \times z)$

  $(x + y) / z = (x / z) + (y / z)$

  $(x - y) / z = (x / z) - (y / z)$

Optimization control – key changes (2)

P 9, 10: ZERO_SUBNORMAL and CONTRACT_OPERATION_CONVERSION apply to the same library functions as the FENV_ROUND and FENV_DEC_ROUND pragmas

- listed functions, where macro replacement is not suppressed (part 1)
- ZERO_SUBNORMAL allows zeroing argument and/or result of sin(subnormal)
- CONTRACT_OPERATION_CONVERSION allows contracting flt_y = sqrt(dbl_x) with fsqrt()

Reproducibility - review

- Support for code sequences whose result values and exception flags are reproducible on any conforming implementation
- #pragma FENV_REPRODUCIBLE *on-off-default*

  FENV_ACCES “on”

  FENV_ALLOW_VALUE_CHANGING_OPTIMIZATION “off”

  FENV_FLT_EVAL_METHOD 0

  FENV_DEC_EVAL_METHOD 1

Reproducibility (2) - review

Rules for reproducible code

- Translates into a sequence of IEC 60559 operations
- Under FENV_REPRODUCIBLE pragma
- Limits use of FP pragmas to reproducible states
- Not use long double, extended floating, complex, or imaginary types
- Use of part 3 interchange formats is reproducible only among supporting implementations

Reproducibility (3) - review

Rules for reproducible code (cont.)

- Not use signaling NaNs
- Not depend on payload or sign bit of quiet NaNs
- Not depend on conversions between floating types and character sequences where character sequences are too long for *correct rounding*
- Etc.

Reproducibility - change

P 13: Clarify that reproducible code does not contain any use that may result in undefined behavior and does not depend on any behavior that is unspecified, implementation-defined, or locale-specific

Alternate exception handling - review

- IEC 60559 default exception handling

  set exception flag(s)

  return prescribed value

  continue execution

- Way for a program to specify alternate exception handling

Alternate exception handling (2) - review

- #pragma STDC FENV_EXCEPT *action except-list*
- *except-list* a comma-separated list of

  exception macro names:

  FE_DIVBYZERO, FE_INVALID, FE_OVERFLOW, …

  FE_ALL_EXCEPT

  optional sub-exception designations:

FE_INVALID_ADD	inf - inf
FE_INVALID_MUL	inf * 0
FE_INVALID_SNAN	signaling NaN operand
FE_DIVBYZERO_LOG	log(0)

  etc.

Alternate exception handling (3) - review

action one of

- DEFAULT

  IEC 60559 default handling

- NO_FLAG

  like default but no flags set

- OPTIONAL_FLAG

  like default but flags may be set

- ABRUPT_UNDERFLOW

  only for “underflow”, IEC 60559-defined abrupt underflow shall occur, unlike ALLOW_ZERO_SUBNORMAL where zeroing may occur

Alternate exception handling - changes

- Optional part of TS 18661-5
- P 2, 15: Separate feature test macro

__STDC_IEC_60559_ATTRIB_ALTERNATE_EXCEPTION_HANDLING__

- Goto actions replaced with try/catch ones
- P 17: Implications of ASAP expanded

Alternate exception handling - changes

action one of (cont.)

- BREAK

  terminate compound statement associated with pragma, ASAP*
- ~~GOTO *label*~~

  ~~jump to labeled statement, ASAP*~~
- ~~DELAYED_GOTO *label*~~

  ~~Complete compound statement associated with pragma, then jump to labeled statement~~

  *ASAP – for performance, the objects, flags, dynamic modes, and library states that would be changed at any point if the compound statement ran to completion are indeterminate or unspecified (P 17)

Alternate exception handling - new

action one of (cont.)

P 18: These work together

- TRY

  A designated exception may be handled (ASAP) by a compound statement associated with a CATCH action

- CATCH

  Code to handle designated exceptions

Alternate exception handling - new

```
double d[n]; float f[n];
...
{
        #pragma STDC FENV_EXCEPT TRY FE_DIVBYZERO,  FE_OVERFLOW
        for (i=0; i<n; 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");
}
```

Alternate exception handling - new

action one of (cont.)

P 18, 19: These work together

- DELAYED_TRY

  After associated compound statement completes, a designated exception may be handled by a compound statement associated with a DELAYED_CATCH action.

- DELAYED_CATCH

  Code to handle designated exceptions

Alternate exception handling - new

Common to ASAP and delayed try/catch …

- IEC 60559 prescribes both
- Catch blocks follow try block
- A catch block is executed only to handle an exception occurring in a try block
- After completion of a catch block execution continues after the last catch block
- No other jumps into or out of try or catch blocks

Alternate exception handling - new

Common to ASAP and delayed try/catch (cont.) …

- A try block shall not be the body of a selection or iteration statement
- … though try and catch blocks together in braces can
- For a catch to handle an exception, one of its exception designation must match one in the try (catch invalid can handle try invalid, but not try all-excepts or try invalid-add)
- An exception designation can appear in at most one catch

Alternate exception handling - new

Differences in ASAP and delayed try/catch …

- Delayed try/catch is deterministic, equivalent to adding code to manage exception flags (P 21)
- ASAP try/catch is not deterministic, for performance – objects, flags, rounding mode, and library state that would be changed at any point if the try block executed to completion are indeterminate or unspecified

Alternate exception handling - new

Differences in ASAP and delayed try/catch (cont.) …

- With delayed try/catch, the jump is to the first catch block with a designation for an occurring exception
- With ASAP try/catch, the jump is to some catch block with a designation for an occurring exception (should be the first occurring exception)
- ASAP try/catch is best implemented by traps, but for most cases can be implemented like delayed try/catch

Alternate exception handling - new

	ASAP	delayed
Input **d**	0.5, 0.0	0.5, 0.0
Results		
f = 1/d	Indeterminate, indeterminate	2, +Infinity
output	"**divide-by-zero**"	"**divide-by-zero**"
"divide-by-zero" flag	Unspecified (set or restored)	Restored (unchanged)
"overflow" flag	Unchanged	Restored (unchanged)

Alternate exception handling - new

	ASAP	delayed
Input **d**	0.5, 1e-100	0.5, 1e-100
Results		
f = 1/d	Indeterminate, Indeterminate	2, +Infinity
output	"**overflow**"	"**overflow**"
"divide-by-zero" flag	Unchanged	Restored (unchanged)
"overflow" flag	Unspecified (set or restored)	Restored

Alternate exception handling - new

	ASAP	delayed
Input **d**	1e-100, 0.0	1e-100, 0.0
Results		
f = 1/d	Indeterminate, Indeterminate	+Infinity, +Infinity
output	"**overflow**" (recommended) or "**divide-by-zero**"	"**divide-by-zero**"
"divide-by-zero" flag	Unspecified (set or restored)	Restored
"overflow" flag	Unspecified (set or restored)	Restored

Alternate exception handling - issues

Should the following be disallowed?

- Currently allow a try without a catch (for delayed try, flags for designated exceptions are restored; for ASAP try, it's unspecified whether flags for designated exceptions are restored)
- Currently allow a catch without a try (it's not executed)
- Currently allow a catch to have an exception designation that does not appear in the try *except-list* (has no effect)

Are *action* names DEFAULT, NO_FLAG, OPTIONAL_FLAG, and BREAK ok?