TS 18661 Part 5
Supplementary attributes

WG 14 N2005
2016-03-07
IEC 60559 attributes

- N2005: draft TS 18661-5 – Supplementary attributes
- Updates N1974 presented in October
- Incorporates input from Oct 2015 WG 14 meeting and review by CFP group and Joseph Myers
- Now ready for PDTS?
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
- Similar 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 (2) – review

• Interaction between evaluation method macros and pragmas
  – Macro values reflect the evaluation method in use, which be might set by a pragma
  – Macros shall not be used in #if and #elif expressions where a pragma is in effect

• _t types have default evaluation formats, but have corresponding type-like macros that (unless undefined) expand to types with the evaluation formats where used
Macro

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

• <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_CHANGEING_OPTIMIZATION allows all the following, which can also be allowed separately
  - ASSOCIATIVE_LAW
  - DISTRIBUTIVE_LAW
  - MULTIPLY_BY_RECIPIROCAL
    \[ A / B = A \times (1/B) \]
Optimization control (2) - review

- **ZERO_SUBNORMAL**
  allow replacing subnormal operands and results with 0
- **CONTRACT_FMA**
  contract (compute with just one rounding) $A \times B + C$
- **CONTRACT_OPERATION_CONVERSION**
  e.g., $F = D_1 \times D_2$ and $F = \sqrt{D}$
- **CONTRACT**
  all contractions
  equivalent to FP_CONTRACT pragma in `<math.h>`
Optimization control (3) – review

• identities derived from allowed identities also allowed, 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 (4) – new

- P 9, 10: ZERO_SUBNORMAL and CONTRACT_OPERATION_CONVERSION affect all applicable library functions for which macro replacement has not been suppressed. Previously said they affected the same functions as constant rounding mode pragmas - but different pragmas don’t all apply to the same functions, e.g., constant rounding mode pragmas don’t affect nextup, but other pragmas do

- P 9: Whether replacing subnormal by zero raises “inexact” and “underflow” is unspecified
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_ACCCES “on”
  FENV_ALLOW_VALUE_CHANGING_OPTIMIZATION “off”
  FENV_FLT_EVAL_METHOD 0
  FENV_DEC_EVAL_METHOD 1
Reproducibility (2) - review

Rules for reproducible code
• Under FENV_REPRODUCIBLE pragma
• Translates into a sequence of IEC 60559 operations
• Not use anything undefined, unspecified, implementation defined, or locale specific
• 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 (4) - new

P 13, 14: Reorganized list of restrictive properties and clarification that they are examples
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

• Optional part of TS 18661-5

• Separate feature test macro

  __STDC_IEC_60559_ATTRIB_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 (4) - review

action one of (cont.)

• BREAK

  terminate compound statement associated with pragma, ASAP*

*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
Alternate exception handling (5) - review

*action* one of (cont.)

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 (6) - review

**action** one of (cont.)

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 (7) – review (except for pragma placement)

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 (8) - review

<table>
<thead>
<tr>
<th></th>
<th>ASAP</th>
<th>delayed</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Input d</strong></td>
<td>0.5, 0.0</td>
<td>0.5, 0.0</td>
</tr>
<tr>
<td><strong>Results</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>f = 1/d</strong></td>
<td>Indeterminate, indeterminate</td>
<td>2, +Infinity</td>
</tr>
<tr>
<td><strong>output</strong></td>
<td>“divide-by-zero”</td>
<td>“divide-by-zero”</td>
</tr>
<tr>
<td><strong>“divide-by-zero” flag</strong></td>
<td>Unspecified (set or restored)</td>
<td>Restored (unchanged)</td>
</tr>
<tr>
<td><strong>“overflow” flag</strong></td>
<td>Unchanged</td>
<td>Restored (unchanged)</td>
</tr>
</tbody>
</table>
## Alternate exception handling (9) - review

<table>
<thead>
<tr>
<th></th>
<th>ASAP</th>
<th>delayed</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Input d</strong></td>
<td>0.5, 1e-100</td>
<td>0.5, 1e-100</td>
</tr>
<tr>
<td><strong>Results</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>f = 1/d</strong></td>
<td>Indeterminate,</td>
<td>2, +Infinity</td>
</tr>
<tr>
<td></td>
<td>Indeterminate</td>
<td></td>
</tr>
<tr>
<td><strong>output</strong></td>
<td>“overflow”</td>
<td>“overflow”</td>
</tr>
<tr>
<td><strong>“divide-by-zero” flag</strong></td>
<td>Unchanged</td>
<td>Restored (unchanged)</td>
</tr>
<tr>
<td><strong>“overflow” flag</strong></td>
<td>Unspecified (set or</td>
<td>Restored</td>
</tr>
<tr>
<td></td>
<td>restored)</td>
<td></td>
</tr>
</tbody>
</table>
# Alternate exception handling (10) - review

<table>
<thead>
<tr>
<th></th>
<th>ASAP</th>
<th>delayed</th>
</tr>
</thead>
<tbody>
<tr>
<td>Input (d)</td>
<td>1e-100, 0.0</td>
<td>1e-100, 0.0</td>
</tr>
<tr>
<td>Results</td>
<td></td>
<td></td>
</tr>
<tr>
<td>(f = 1/d)</td>
<td>Indeterminate, Indeterminate</td>
<td>+Infinity, +Infinity</td>
</tr>
<tr>
<td>output</td>
<td>“overflow” (recommended)</td>
<td>“divide-by-zero”</td>
</tr>
<tr>
<td></td>
<td>or “divide-by-zero”</td>
<td></td>
</tr>
<tr>
<td>“divide-by-zero” flag</td>
<td>Unspecified (set or restored)</td>
<td>Restored</td>
</tr>
<tr>
<td>“overflow” flag</td>
<td>Unspecified (set or restored)</td>
<td>Restored</td>
</tr>
</tbody>
</table>
Alternate exception handling (11) - review

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 (12) - review

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 (13) - review

Differences in ASAP and delayed try/catch ...

• Delayed try/catch is deterministic, equivalent to adding code to manage exception flags

• 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 (14) - review

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 (15) - new

- P 15: Designations in except-list must be distinct
- P 15: Sub-exception designations are macros suitable for feature tests
- P 18: Try exception designations must appear in one and only one catch
- P 17, 20, 22, 23: Pragmas for control flow actions appear before (not in) their associated blocks
Alternate exception handling (16) - new

- P 15: Clarification that supported sub-exceptions occur only as specified
- P 19: FENV_EXACT pragmas apply to all applicable library functions whose macro expansion has not been suppressed
- P 19: Interaction with signal handling leads to undefined behavior
- P 22: Example with overlapping exception designations