ISO/IEC JTC 1/SC 22/WG 14 N1919

Date: yyyy-mm-dd

Reference number of document: **ISO/IEC TS 18661-5**

Committee identification: ISO/IEC JTC 1/SC 22/WG 14

Secretariat: ANSI

Information technology — Programming languages, their environments, and system software interfaces — Floating-point extensions for C — Part 5: Supplementary attributes

Technologies de l'information — Langages de programmation, leurs environnements et interfaces du logiciel système — Extensions à virgule flottante pour C — Partie 5: Attributs supplémentaires

Document type: Technical Specification
Document subtype:
Document stage: (20) Preparation
Document language: E

Contents

Page

Foreword

ISO (the International Organization for Standardization) and IEC (the International Electrotechnical Commission) form the specialized system for worldwide standardization. National bodies that are members of ISO or IEC participate in the development of International Standards through technical committees established by the respective organization to deal with particular fields of technical activity. ISO and IEC technical committees collaborate in fields of mutual interest. Other international organizations, governmental and non-governmental, in liaison with ISO and IEC, also take part in the work. In the field of information technology, ISO and IEC have established a joint technical committee, ISO/IEC JTC 1.

The procedures used to develop this document and those intended for its further maintenance are described in the ISO/IEC Directives, Part 1. In particular the different approval criteria needed for the different types of document should be noted. This document was drafted in accordance with the editorial rules of the ISO/IEC Directives, Part 2 (see www.iso.org/directives).

Attention is drawn to the possibility that some of the elements of this document may be the subject of patent rights. ISO and IEC shall not be held responsible for identifying any or all such patent rights. Details of any patent rights identified during the development of the document will be in the Introduction and/or on the ISO list of patent declarations received (see www.iso.org/patents).

Any trade name used in this document is information given for the convenience of users and does not constitute an endorsement.

For an explanation on the meaning of ISO specific terms and expressions related to conformity assessment, as well as information about ISO's adherence to the WTO principles in the Technical Barriers to Trade (TBT) see the following URL: Foreword - Supplementary information

The committee responsible for this document is ISO/IEC JTC 1, *Information technology*, SC 22, *Programming languages, their environments, and system software interfaces*.

ISO/IEC TS 18661 consists of the following parts, under the general title *Information technology— Programming languages, their environments, and system software interfaces — Floating-point extensions for C*:

— *Part 1: Binary floating-point arithmetic*

— *Part 2: Decimal floating-point arithmetic*

— *Part 3: Interchange and extended types*

— *Part 4: Supplementary functions*

— *Part 5: Supplementary attributes*

Part 1 updates ISO/IEC 9899:2011, *Information technology — Programming Language C*, Annex F in particular, to support all required features of ISO/IEC/IEEE 60559:2011, *Information technology — Microprocessor Systems — Floating-point arithmetic*.

Part 2 supersedes ISO/IEC TR 24732:2009, *Information technology — Programming languages, their environments and system software interfaces — Extension for the programming language C to support decimal floating-point arithmetic*.

Parts 3-5 specify extensions to ISO/IEC 9899:2011 for features recommended in ISO/IEC/IEEE 60559:2011.

Introduction

Background

IEC 60559 floating-point standard

The IEEE 754-1985 standard for binary floating-point arithmetic was motivated by an expanding diversity in floating-point data representation and arithmetic, which made writing robust programs, debugging, and moving programs between systems exceedingly difficult. Now the great majority of systems provide data formats and arithmetic operations according to this standard. The IEC 60559:1989 international standard was equivalent to the IEEE 754-1985 standard. Its stated goals were:

1 Facilitate movement of existing programs from diverse computers to those that adhere to this standard.

2 Enhance the capabilities and safety available to programmers who, though not expert in numerical methods, may well be attempting to produce numerically sophisticated programs. However, we recognize that utility and safety are sometimes antagonists.

3 Encourage experts to develop and distribute robust and efficient numerical programs that are portable, by way of minor editing and recompilation, onto any computer that conforms to this standard and possesses adequate capacity. When restricted to a declared subset of the standard, these programs should produce identical results on all conforming systems.

4 Provide direct support for

   a. Execution-time diagnosis of anomalies

   b. Smoother handling of exceptions

   c. Interval arithmetic at a reasonable cost

5 Provide for development of

   a. Standard elementary functions such as exp and cos

   b. Very high precision (multiword) arithmetic

   c. Coupling of numerical and symbolic algebraic computation

6 Enable rather than preclude further refinements and extensions.

To these ends, the standard specified a floating-point model comprising:

formats – for binary floating-point data, including representations for Not-a-Number (NaN) and signed infinities and zeros

operations – basic arithmetic operations (addition, multiplication, etc.) on the format data to compose a well-defined, closed arithmetic system; also specified conversions between floating-point formats and decimal character sequences, and a few auxiliary operations

context – status flags for detecting exceptional conditions (invalid operation, division by zero, overflow, underflow, and inexact) and controls for choosing different rounding methods

The ISO/IEC/IEEE 60559:2011 international standard is equivalent to the IEEE 754-2008 standard for floating-point arithmetic, which is a major revision to IEEE 754-1985.

The revised standard specifies more formats, including decimal as well as binary. It adds a 128-bit binary format to its basic formats. It defines extended formats for all of its basic formats. It specifies data interchange formats (which may or may not be arithmetic), including a 16-bit binary format and an unbounded tower of wider formats. To conform to the floating-point standard, an implementation must provide at least one of the basic formats, along with the required operations.

The revised standard specifies more operations. New requirements include – among others – arithmetic operations that round their result to a narrower format than the operands (with just one rounding), more conversions with integer types, more classifications and comparisons, and more operations for managing flags and modes. New recommendations include an extensive set of mathematical functions and seven reduction functions for sums and scaled products.

The revised standard places more emphasis on reproducible results, which is reflected in its standardization of more operations. For the most part, behaviors are completely specified. The standard requires conversions between floating-point formats and decimal character sequences to be correctly rounded for at least three more decimal digits than is required to distinguish all numbers in the widest supported binary format; it fully specifies conversions involving any number of decimal digits. It recommends that transcendental functions be correctly rounded.

The revised standard requires a way to specify a constant rounding direction for a static portion of code, with details left to programming language standards. This feature potentially allows rounding control without incurring the overhead of runtime access to a global (or thread) rounding mode.

Other features recommended by the revised standard include alternate methods for exception handling, controls for expression evaluation (allowing or disallowing various optimizations), support for fully reproducible results, and support for program debugging.

The revised standard, like its predecessor, defines its model of floating-point arithmetic in the abstract. It neither defines the way in which operations are expressed (which might vary depending on the computer language or other interface being used), nor does it define the concrete representation (specific layout in storage, or in a processor's register, for example) of data or context, except that it does define specific encodings that are to be used for data that may be exchanged between different implementations that conform to the specification.

IEC 60559 does not include bindings of its floating-point model for particular programming languages. However, the revised standard does include guidance for programming language standards, in recognition of the fact that features of the floating-point standard, even if well supported in the hardware, are not available to users unless the programming language provides a commensurate level of support. The implementation’s combination of both hardware and software determines conformance to the floating-point standard.

C support for IEC 60559

The C standard specifies floating-point arithmetic using an abstract model. The representation of a floating-point number is specified in an abstract form where the constituent components (sign, exponent, significand) of the representation are defined but not the internals of these components. In particular, the exponent range, significand size, and the base (or radix) are implementation-defined. This allows flexibility for an implementation to take advantage of its underlying hardware architecture. Furthermore, certain behaviors of operations are also implementation-defined, for example in the area of handling of special numbers and in exceptions.

The reason for this approach is historical. At the time when C was first standardized, before the floating-point standard was established, there were various hardware implementations of floating-point arithmetic in common use. Specifying the exact details of a representation would have made most of the existing implementations at the time not conforming.

Beginning with ISO/IEC 9899:1999 (C99), C has included an optional second level of specification for implementations supporting the floating-point standard. C99, in conditionally normative Annex F, introduced nearly complete support for the IEC 60559:1989 standard for binary floating-point arithmetic. Also, C99’s informative Annex G offered a specification of complex arithmetic that is compatible with IEC 60559:1989.

ISO/IEC 9899:2011 (C11) includes refinements to the C99 floating-point specification, though is still based on IEC 60559:1989. C11 upgrades Annex G from “informative” to “conditionally normative”.

ISO/IEC TR 24732:2009 introduced partial C support for the decimal floating-point arithmetic in ISO/IEC/IEEE 60559:2011. ISO/IEC TR 24732, for which technical content was completed while IEEE 754-2008 was still in the later stages of development, specifies decimal types based on ISO/IEC/IEEE 60559:2011 decimal formats, though it does not include all of the operations required by ISO/IEC/IEEE 60559:2011.

Purpose

The purpose of ISO/IEC TS 18661 is to provide a C language binding for ISO/IEC/IEEE 60559:2011, based on the C11 standard, that delivers the goals of ISO/IEC/IEEE 60559 to users and is feasible to implement. It is organized into five Parts.

Part 1 provides changes to C11 that cover all the requirements, plus some basic recommendations, of ISO/IEC/IEEE 60559:2011 for binary floating-point arithmetic. C implementations intending to support ISO/IEC/IEEE 60559:2011 are expected to conform to conditionally normative Annex F as enhanced by the changes in Part 1.

Part 2 enhances ISO/IEC TR 24732 to cover all the requirements, plus some basic recommendations, of ISO/IEC/IEEE 60559:2011 for decimal floating-point arithmetic. C implementations intending to provide an extension for decimal floating-point arithmetic supporting ISO/IEC/IEEE 60559:2011 are expected to conform to Part 2.

Part 3 specifies types and other support for interchange and extended formats recommended in ISO/IEC/IEEE 60559:2011. C implementations intending to provide an extension for these formats are expected to conform to Part 3.

Part 4 specifies functions for operations recommended in ISO/IEC/IEEE 60559:2011. C implementations intending to provide an extension for these operations are expected to conform to Part 4.

Part 5, this document, specifies support for attributes recommended in ISO/IEC/IEEE 60559:2011. C implementations intending to provide an extension for these attributes are expected to conform to Part 5.

Additional background on supplementary attributes

ISO/IEC/IEEE 60559:2011 defines alternatives for certain attributes of floating-point semantics and intends that programming languages provide means by which a program can specify which of the alternative semantics apply to a given block of code. The program specification of attributes is to be constant (fixed at translation time), not dynamic (changeable at execution time).

ISO/IEC TS 18661 provides these attributes by means of standard pragmas, where the pragma parameters represent the alternative semantics.

The **`FENV_ROUND`** and **`FENV_DEC_ROUND`** pragmas, specified in ISO/IEC TS 18661-1 and ISO/IEC TS 18661-2, respectively, provide the rounding direction attributes required by ISO/IEC/IEEE 60559:2011.

ISO/IEC/IEEE 60559:2011 recommends attributes for

— preferredWidth: evaluation formats for floating-point operations

— value-changing optimization: allow/disallow program transformations that might affect floating-point result values

— alternate exception handling: methods of handling floating-point exceptions

— reproducibility: support for getting floating-point result values and exceptions that are exactly reproducible on other systems

This part of ISO/IEC TS 18661 specifies pragmas that provide these attributes.

Note that the pragmas introduced by ISO/IEC TS 18661 are similar to the floating-point pragmas (**`FENV_ACCESS`**, **`FP_CONTRACT`**, **`CX_LIMITED_RANGE`**) that are already in ISO/IEC 9899. They all have the same general syntactic form, usage requirements, and range of effect.

Information technology — Programming languages, their environments, and system software interfaces — Floating-point extensions for C —

Part 5:

Supplementary attributes

1 Scope

This part of ISO/IEC TS 18661 extends programming language C to include support for attributes specified and recommended in ISO/IEC/IEEE 60559:2011.

2 Conformance

An implementation conforms to this part of ISO/IEC TS 18661 if

a) It meets the requirements for a conforming implementation of C11 with all the changes to C11 as specified in parts 1-5 of ISO/IEC TS 18661;

b) It conforms to ISO/IEC TS 18661-1 or ISO/IEC TS 18661-2 (or both); and

c) It defines `__STDC_IEC_60559_ATTRIBS__` to `201`*ymm*`L`.

3 Normative references

The following referenced documents are indispensable for the application of this document. Only the editions cited apply.

ISO/IEC 9899:2011, *Information technology — Programming languages, their environments and system software interfaces — Programming Language C*

ISO/IEC 9899:2011/Cor.1:2012, *Technical Corrigendum 1*

ISO/IEC/IEEE 60559:2011, *Information technology — Microprocessor Systems — Floating-point arithmetic* (with identical content to IEEE 754-2008, *IEEE Standard for Floating-Point Arithmetic*. The Institute of Electrical and Electronic Engineers, Inc., New York, 2008)

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

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

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

Changes specified in this part of ISO/IEC TS 18661 are relative to ISO/IEC 9899:2011, including *Technical Corrigendum 1* (ISO/IEC 9899:2011/Cor. 1:2012), together with the changes from parts 1-4 of ISO/IEC TS 18661.

4 Terms and definitions

For the purposes of this document, the terms and definitions given in ISO/IEC 9899:2011, ISO/IEC/IEEE 60559:2011, parts 1-4 of ISO/IEC TS 18661, and the following apply.

4.1
C11
standard ISO/IEC 9899:2011, *Information technology — Programming languages, their environments and system software interfaces — Programming Languages — C*, including *Technical Corrigendum 1* (ISO/IEC 9899:2011/Cor. 1:2012)

5 C standard conformance

5.1 Freestanding implementations

The specification in C11 + TS18661-1 + TS18661-2 allows freestanding implementations to conform to this part of Technical Specification 18661.

5.2 Predefined macros

Change to C11 + TS18661-1 + TS18661-2 + TS18661-3 + TS18661-4:

In 6.10.8.3#1, add:

`__STDC_IEC_60559_ATTRIBS__` The integer constant **`201ymmL`**, intended to indicate support of attributes specified and recommended in IEC 60559.

6 Standard pragmas

C11 provides standard pragmas (6.10.6) for specifying certain attributes pertaining to floating-point behavior within a block or file. This part of ISO/IEC TS 16881 extends this practice by introducing additional standard pragmas to support the attributes recommended by IEC 60559.

Change to C11 + TS18661-1 + TS18661-2 + TS18661-3 + TS18661-4:

In 6.10.6#2, append to the list of standard pragmas:

```
#pragma STDC FENV_FLT_EVAL_METHOD width
#pragma STDC FENV_DEC_EVAL_METHOD width
#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
#pragma STDC FENV_EXCEPT except-list action
#pragma STDC FENV_REPRODUCIBLE on-off-switch
```

width: specified with the pragmas (7.6.1c, 7.6.1d)

except-list, *action*: specified with the pragmas (7.6.1e.1)

7 Evaluation formats

IEC 60559 recommends attributes for specifying a preferred width for operation results. These preferred widths correspond to the evaluation formats defined in C11, though C11 does not provide means for the user to control the evaluation format. This part of ISO/IEC TS 16881 provides pragmas in **`<fenv.h>`** to control the evaluation format, using the values of the **`FLT_EVAL_METHOD`** and **`DEC_EVAL_METHOD`** macros (5.2.4.2.2a) to represent the evaluation formats.

Change to C11 + TS18661-1 + TS18661-2 + TS18661-3 + TS18661-4:

After 7.6.1b, insert:

7.6.1c Evaluation method pragma

Synopsis

[1]
```
#define __STDC_WANT_IEC_60559_ATTRIBS_EXT__
#include <fenv.h>
#pragma STDC FENV_FLT_EVAL_METHOD width
```

Description

[2] The **`FENV_FLT_EVAL_METHOD`** pragma sets the evaluation method for standard floating types and for binary interchange and extended floating types to the evaluation method represented by *width*, where *width* is an integer value specified in 5.2.4.2.2a as a possible value for the **`FLT_EVAL_METHOD`** macro. The effect of the pragma is as specified in 5.2.4.2.2a. *width* may be **`-1`**, **`0`**, or the value of **`FLT_EVAL_METHOD`**. Which, if any, other values of *width* are supported is implementation-defined. Use of unsupported values of *width* results in undefined behavior. The pragma shall occur either outside external declarations or preceding all explicit declarations and statements inside a compound statement. When outside external declarations, the pragma takes effect from its occurrence until another **`FENV_FLT_EVAL_METHOD`** pragma is encountered, or until the end of the translation unit. When inside a compound statement, the pragma takes effect from its occurrences until another **`FENV_FLT_EVAL_METHOD`** pragma is encountered (including within a nested compound statement), or until the end of the compound statement.

7.6.1d Evaluation method pragma for decimal

Synopsis

[1]
```
#define __STDC_WANT_IEC_60559_DFP_EXT__
#define __STDC_WANT_IEC_60559_ATTRIBS_EXT__
#include <fenv.h>
#pragma STDC FENV_DEC_EVAL_METHOD width
```

Description

[2] The **`FENV_DEC_EVAL_METHOD`** pragma sets the evaluation method for decimal interchange and extended floating types to the evaluation method represented by *width*, where *width* is an integer value specified in 5.2.4.2.2a as a possible value for the **`DEC_EVAL_METHOD`** macro. The effect of the pragma is as specified in 5.2.4.2.2a. *width* may be **`-1`**, **`1`**, or the value of **`DEC_EVAL_METHOD`**. Which, if any, other values of *width* are supported is implementation-defined. Use of unsupported values of *width* results in undefined behavior. The pragma shall occur either outside external declarations or preceding all explicit declarations and statements inside a compound statement. When outside external declarations, the pragma takes effect from its occurrence until another **`FENV_DEC_EVAL_METHOD`** pragma is encountered, or until the end of the translation unit. When inside a compound statement, the pragma takes effect from its occurrences until another **`FENV_DEC_EVAL_METHOD`** pragma is encountered (including within a nested compound statement), or until the end of the compound statement.

8 Optimization controls

IEC 60559 recommends attributes to allow and disallow value-changing optimizations, individually and collectively. C11 Annex F disallows value-changing optimizations, except for contractions which can be controlled as a group with the `FP_CONTRACT` pragma. This part of ISO/IEC TS 18661 provides pragmas to allow or disallow certain value-changing optimizations, including those mentioned in IEC 60559.

Change to C11 + TS18661-1 + TS18661-2 + TS18661-3 + TS18661-4:

After 7.6.1d, insert:

7.6.1e Optimization control pragmas

[1] The pragmas in this subclause can be used to allow or disallow program transformations, here referred to as *optimizations*, that might change the values of floating-point expressions. They apply to all floating-point types. It is unspecified whether an optimization allowed by these pragmas actually occurs consistently, or at all.

[2] Each pragma shall occur either outside external declarations or preceding all explicit declarations and statements inside a compound statement. When outside external declarations, the pragma takes effect, on each optimization it controls, from its occurrence until another pragma that affects the same optimization is encountered, or until the end of the translation unit. When inside a compound statement, the pragma takes effect, on each optimization it controls, from its occurrence until another pragma that affects the same optimization is encountered (including within a nested compound statement), or until the end of the compound statement.

7.6.1e.1 The `FENV_ALLOW_VALUE_CHANGING_OPTIMIZATION` pragma

Synopsis

[1]
```
#define __STDC_WANT_IEC_60559_ATTRIBS_EXT__
#include <fenv.h>
#pragma STDC FENV_ALLOW_VALUE_CHANGING_OPTIMIZATION on-off-switch
```

Description

[1] This pragma is equivalent to all the optimization pragmas specified below, with the same value of *on-off-switch* (`ON`, `OFF`, or `DEFAULT`).

[2] NOTE The `FENV_ALLOW_VALUE_CHANGING_OPTIMIZATION` pragma does not affect the evaluation methods. Nevertheless, an evaluation method characterized by a negative value of *width* (5.2.4.2.2a) might allow for indeterminable evaluation formats, hence unspecified result values.

7.6.1e.2 The `FENV_ALLOW_ASSOCIATIVE_LAW` pragma

Synopsis

[1]
```
#define __STDC_WANT_IEC_60559_ATTRIBS_EXT__
#include <fenv.h>
#pragma STDC FENV_ALLOW_ASSOCIATIVE_LAW on-off-switch
```

Description

[2] This pragma allows or disallows optimization based on the associative laws for addition and multiplication

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

where *on-off-switch* is one of

> **`ON`** – allow application of the associative laws
>
> **`OFF`** – do not allow application of the associative laws
>
> **`DEFAULT`** – “off”

7.6.1e.3 The `FENV_ALLOW_DISTRIBUTIVE_LAW` pragma

Synopsis

[1]
```
#define __STDC_WANT_IEC_60559_ATTRIBS_EXT__
#include <fenv.h>
#pragma STDC FENV_ALLOW_DISTRIBUTIVE_LAW on-off-switch
```

Description

[2] This pragma allows or disallows optimization based on the distributive laws for multiplication and division

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

where *on-off-switch* is one of

> **`ON`** – allow application of the distributive laws
>
> **`OFF`** – do not allow application of the distributive laws
>
> **`DEFAULT`** – “off”

7.6.1e.4 The `FENV_ALLOW_MULTIPLY_BY_RECIPROCAL` pragma

Synopsis

[1]
```
#define __STDC_WANT_IEC_60559_ATTRIBS_EXT__
#include <fenv.h>
#pragma STDC FENV_ALLOW_MULTIPLY_BY_RECIPROCAL on-off-switch
```

Description

[2] This pragma allows or disallows optimization based on the mathematical equivalence of division and multiplication by the reciprocal of the denominator

$x / y = x \times (1 / y)$

where *on-off-switch* is one of

> **`ON`** – allow multiply by reciprocal
>
> **`OFF`** – do not allow multiply by reciprocal
>
> **`DEFAULT`** – “off”

7.6.1e.5 The `FENV_ALLOW_ZERO_SUBNORMAL` pragma

Synopsis

[1]
```
#define __STDC_WANT_IEC_60559_ATTRIBS_EXT__
#include <fenv.h>
#pragma STDC FENV_ALLOW_ZERO_SUBNORMAL on-off-switch
```

Description

[2] This pragma allows or disallows replacement of subnormal operands and results by zero, where *on-off-switch* is one of

`ON` – allow replacement of subnormals with zero

`OFF` – do not allow replacement of subnormals with zero

`DEFAULT` – “off”

7.6.1e.6 The `FENV_ALLOW_CONTRACT_FMA` pragma

Synopsis

[1]
```
#define __STDC_WANT_IEC_60559_ATTRIBS_EXT__
#include <fenv.h>
#pragma STDC FENV_ALLOW_CONTRACT_FMA on-off-switch
```

Description

[2] This pragma allows or disallows contraction (6.5) of floating-point multiply and add (to the result of the multiply)

x * *y* + *z*

where *on-off-switch* is one of

`ON` – allow contraction for floating-point multiply-add

`OFF` – do not allow contraction for floating-point multiply-add

`DEFAULT` – implementation defined whether “on” or “off”

[3] NOTE IEC 60559 uses the term *synthesize* instead of *contract*.

7.6.1e.7 The `FENV_ALLOW_CONTRACT_OPERATION_CONVERSION` pragma

Synopsis

[1]
```
#define __STDC_WANT_IEC_60559_ATTRIBS_EXT__
#include <fenv.h>
#pragma STDC FENV_ALLOW_CONTRACT_OPERATION_CONVERSION on-off-switch
```

Description

[2] This pragma allows or disallows contraction (6.5) of a floating-point operation and a conversion (of the result of the operation), where *on-off-switch* is one of

`ON` – allow contraction for floating-point operation-conversion

`OFF` – do not allow contraction for floating-point operation-conversion

`DEFAULT` – implementation defined whether “on” or “off”

[3] EXAMPLE For the code sequence

```
#define __STDC_WANT_IEC_60559_ATTRIBS_EXT__
#include <fenv.h>
#pragma STDC FENV_ALLOW_CONTRACT_OPERATION_CONVERSION ON
float a;
double b, c;
…
a = b * c;
```

the multiply (operation) and assignment (conversion) are allowed to be evaluated with just one rounding (to the range and precision of **`float`**). If the *on-off-switch* for the pragma were **`OFF`**, then the multiply would have to be rounded according to the evaluation method and the assignment would require a second rounding.

7.6.1e.8 The `FENV_ALLOW_CONTRACT` pragma

Synopsis

[1]
```
#define __STDC_WANT_IEC_60559_ATTRIBS_EXT__
#include <fenv.h>
#pragma STDC FENV_ALLOW_CONTRACT on-off-switch
```

Description

[2] This pragma allows or disallows contraction (6.5) for floating-point operations, where *on-off-switch* is one of

`ON` – allow contraction for floating-point operations

`OFF` – do not allow contraction for floating-point operations

`DEFAULT` – implementation defined whether “on” or “off”

[3] The optimizations controlled by this pragma include those controlled by the **`FENV_ALLOW_CONTRACT_FMA`** and **`FENV_ALLOW_CONTRACT_OPERATION_CONVERSION`** pragmas.

[4] This pragma is equivalent to the **`FP_CONTRACT`** pragma in **`<math.h>`**: the two pragmas may be used interchangeably, provided the appropriate header is included and the implementation defines **`__STDC_WANT_IEC_60559_ATTRIBS_EXT__`**.

9 Alternate exception handling

THIS CLAUSE IS STILL IN PROGRESS

10 Reproducibility

IEC 60559 recommends an attribute to facilitate writing programs whose floating-point results and exception flags will be reproducible on any implementation that supports the language and library features used by the program. Such code must use only those features of the language and library that support reproducible results. These features include ones with a well-defined binding to reproducible features of IEC 60559, so that no unspecified or implementation-defined behavior is admitted. This part of ISO/IEC TS 18661 provides a pragma to support the IEC 60559 attribute for reproducible results and gives requirements for programs to have reproducible results.

Change to C11 + TS18661-1 + TS18661-2 + TS18661-3 + TS18661-4:

After 7.6.1e, insert:

7.6.1f Reproducible results

[1] The pragma in this subclause supports the reproducible results attribute recommended in IEC 60559. Where the state of the pragma is "on", floating-point numerical results and exception flags are reproducible on implementations that define **`__STDC_IEC_60559_ATTRIBS__`** and that support the language and library features used by the source code, provided the source code uses a limited set of features as described below (7.6.1f.2).

[2] An implementation that defines **`__STDC_IEC_60559_ATTRIBS__`** also defines either **`__STDC_IEC_60559_BFP__`** or **`__STDC_IEC_60559_DFP__`**, or both. If the implementation defines **`__STDC_IEC_60559_BFP__`**, it supports reproducible results for binary floating-point arithmetic. If the implementation defines **`__STDC_IEC_60559_DFP__`**, it supports reproducible results for decimal floating-point arithmetic. If the implementation defines **`__STDC_IEC_60559_TYPES__`**, then it supports reproducible results for code using its interchange floating types. If the implementation defines **`__STDC_IEC_60559_FUNCS__`** and it provides a set of correctly rounded math functions (7.31.6a), then it supports reproducible results for code using correctly rounded math functions from that set.

7.6.1f.1 The `FENV_REPRODUCIBLE` pragma

Synopsis

[1]
```
#define __STDC_WANT_IEC_60559_ATTRIBS_EXT__
#include <fenv.h>
#pragma STDC FENV_REPRODUCIBLE on-off-switch
```

Description

[2] This pragma enables or disables support for reproducible results. The pragma shall occur either outside external declarations or preceding all explicit declarations and statements inside a compound statement. When outside external declarations, the pragma takes effect from its occurrence until another **`FENV_REPRODUCIBLE`** pragma is encountered, or until the end of the translation unit. When inside a compound statement, the pragma takes effect from its occurrence until another **`FENV_REPRODUCIBLE`** pragma is encountered (including within a nested compound statement), or until the end of the compound statement; at the end of a compound statement the state for the pragma is restored to its condition just before the compound statement.

[3] If the state of the pragma is "on", then the effects of the following are implied

```
#pragma STDC FENV_ACCESS ON
#pragma STDC FENV_ALLOW_VALUE_CHANGING_OPTIMIZATION OFF
```

and if **`__STDC_IEC_60559_BFP__`** is defined

```
#pragma STDC FENV_FLT_EVAL_METHOD 0
```

and if `__STDC_IEC_60559_DFP__` is defined

```
#pragma STDC FENV_DEC_EVAL_METHOD 1
```

[4] If the pragma appears with the *on-off-switch* **OFF** under the effect of a pragma with *on-off-switch* **ON**, then the states of the **FENV_ACCESS** pragma, the value-changing optimization pragmas, and the evaluation method pragmas (even an evaluation method pragma whose state was explicitly changed under the effect of the pragma with *on-off-switch* **ON**) revert to their states prior to the pragma with on-off-switch **ON**. The pragma with *on-off-switch* **OFF** has no effect if it occurs where the state of the pragma is "off".

[5] The "default" state of the pragma is "off".

[6] The implementation should produce a diagnostic message if, where the state of the **FENV_REPRODUCIBLE** pragma is "on", the source code uses a language or library feature whose results may not be reproducible.

7.6.1f.2 Reproducible code

[1] Following are requirements for a code sequence in order that its results will be reproducible. If the code uses optional features noted below, then results are reproducible only on implementations that support those features.

— The code translates into a sequence of floating-point operations that are bound to IEC 60559 operations, as described in F.3 in the table entitled "Operation binding". Code using other C features must not depend on unspecified, undefined, or implementation-defined behaviors listed in Annex J.

— The code is under the effect of the **FENV_REPRODUCIBLE** pragma (with state "on").

— The code does not set the state of any pragma that allows value-changing optimizations to "on" or "default".

— The code does not set the state of the **FENV_ACCESS** pragma to "off" or "default".

— The code does not use the **FENV_FLT_EVAL_METHOD** pragma with any *width* except 0 or 1. Support for *width* equal to 1 is an optional feature.

— The code does not use the **FENV_DEC_EVAL_METHOD** pragma with any *width* except 1 or 2. Support for *width* equal to 2 is an optional feature.

— The code does not set the state of the **FENV_ROUND** or **FENV_DEC_ROUND** pragmas to any implementation-defined state.

— Use of an **FENV_EXCEPT** pragma with an *except-list* that includes sub-exceptions is an optional feature.

— The code does not use an **FENV_EXCEPT** pragma with an *action* **BREAK** or **GOTO**.

— Where the state of the **FENV_EXCEPT** pragma is "on", the code does not depend on the floating-point exceptions specified by *except-list* if *action* is **NOEXCEPT** or **OPTEXCEPT**.

— The code does not use the **long double** type.

— If **`__STDC_IEC_60559_BFP__`** is not defined by the implementation, the code does not use any standard floating types.

— Even if **`__STDC_IEC_60559_TYPES__`** is defined, the code does not use extended floating types. Even if **`__STDC_IEC_60559_TYPES__`** is defined, some interchange floating types are optional features.

— The code does not use complex or imaginary types.

— The code does not use signaling NaNs.

— The code does not use the **`remquo`** functions.

— The code does not depend on the “invalid” floating-point exception from the **`fma`** functions, or the functions that compute multiply-add rounded to narrower type, when the product factors are zero and infinite and the term added to the product is a quiet NaN.

— The code does not depend on the sign of a zero result or the quantum of a decimal result for the **`fmin`**, **`fmax`**, **`fminmag`**, and **`fmaxmag`** functions when the arguments are equal.

— The code does not depend on the payloads or sign bits of quiet NaNs.

— The code does not depend on the “underflow” or “inexact” floating-point exceptions.

— The code does not use extended integer types.

— The code does not depend on the return value of operations (implicit or explicit) that return a value in integer format when the operation raises the “invalid” floating-point exception.

— The code does not depend on conversions between binary floating types and character sequences with more than *M* + 3 significant decimal digits, where *M* is 17 if **`__STDC_IEC_60559_TYPES__`** is not defined (by the implementation), and *M* is $1 + \lceil p \times \log_{10}(2) \rceil$, where *p* is the precision of the widest supported binary interchange floating type, if **`__STDC_IEC_60559_TYPES__`** is defined. Even if **`__STDC_IEC_60559_TYPES__`** is defined, support for interchange floating types wider than binary64 is an optional feature. (This specification differs from IEC 60559 which requires that an implementation supporting reproducibility not limit the number of significant decimal digits for correct rounding.)

— The code does not depend on the actual character sequence in **`printf`** results with style **`a`** (or **`A`**), nor does it depend on numerical values of such results when the precision is not sufficient for an exact representation.

Bibliography

[1] ISO/IEC TR 24732:2009, *Information technology – Programming languages, their environments and system software interfaces – Extension for the programming language C to support decimal floating-point arithmetic*

[2] IEC 60559:1989, *Binary floating-point arithmetic for microprocessor systems, second edition*

[3] IEEE 754-2008, *IEEE Standard for Floating-Point Arithmetic*

[4] IEEE 754−1985, *IEEE Standard for Binary Floating-Point Arithmetic*

[5] IEEE 854−1987, *IEEE Standard for Radix-Independent Floating-Point Arithmetic*