Technical Specification

ISO/IEC TS 18661-5

Second edition
2025-03

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

Part 5: **Supplementary attributes**

Langages de programmation, leurs environnements et interfaces du logiciel système — Extensions à virgule flottante pour C —

Partie 5: Attributs supplémentaires

STANDARDSISO.COM : Click to view the full PDF of ISO/IEC TS 18661-5:2025

Reference number
ISO/IEC TS 18661-5:2025(en)

© ISO/IEC 2025

STANDARDSISO.COM : Click to view the full PDF of ISO/IEC TS 18661-5:2025

COPYRIGHT PROTECTED DOCUMENT

© ISO/IEC 2025

All rights reserved. Unless otherwise specified, or required in the context of its implementation, no part of this publication may be reproduced or utilized otherwise in any form or by any means, electronic or mechanical, including photocopying, or posting on the internet or an intranet, without prior written permission. Permission can be requested from either ISO at the address below or ISO's member body in the country of the requester.

ISO copyright office
CP 401 • Ch. de Blandonnet 8
CH-1214 Vernier, Geneva
Phone: +41 22 749 01 11
Email: copyright@iso.org
Website: www.iso.org

Published in Switzerland

Contents

STANDARDSISO.COM : Click to view the full PDF of ISO/IEC TS 18661-5:2025

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.

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 or www.iec.ch/members_experts/refdocs).

ISO and IEC draw attention to the possibility that the implementation of this document may involve the use of (a) patent(s). ISO and IEC take no position concerning the evidence, validity or applicability of any claimed patent rights in respect thereof. As of the date of publication of this document, ISO and IEC had not received notice of (a) patent(s) which may be required to implement this document. However, implementers are cautioned that this may not represent the latest information, which may be obtained from the patent database available at www.iso.org/patents and https://patents.iec.ch. ISO and IEC shall not be held responsible for identifying any or all such patent rights.

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

For an explanation of the voluntary nature of standards, the meaning of ISO specific terms and expressions related to conformity assessment, as well as information about ISO's adherence to the World Trade Organization (WTO) principles in the Technical Barriers to Trade (TBT) see www.iso.org/iso/foreword.html. In the IEC, see www.iec.ch/understanding-standards.

This document was prepared by Joint Technical Committee ISO/IEC JTC 1, *Information technology*, Subcommittee SC 22, *Programming languages, their environments and system software interfaces*.

This second edition cancels and replaces the first edition (ISO/IEC TS 18661-5:2016), which has been technically revised.

The main changes are as follows:

— The specification has been updated to extend ISO/IEC 9899:2024.

— Conformance macros have been added to allow conformance to each of the four feature sets (evaluation formats, optimization controls, reproducibility, and alternate exception handling) independently.

A list of all parts in the ISO/IEC 18661 series can be found on the ISO website.

Any feedback or questions on this document should be directed to the user's national standards body. A complete listing of these bodies can be found at www.iso.org/members.html and www.iec.ch/national-committees.

STANDARDSISO.COM : Click to view the full PDF of ISO/IEC TS 18661-5:2025

Introduction

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 reliable programs, debugging and moving programs between systems exceedingly difficult. Now the great majority of systems provide data formats and arithmetic operations according to IEEE 754. Corresponding versions of IEEE 754 and ISO/IEC 60559 have equivalent content.

Support for IEEE 754-1985 was added in ISO/IEC 9899:1999 (also referred to as C99), and ISO/IEC 9899:2018 is still based on IEEE 754-1985. However, IEEE 754 underwent a major revision in 2008 and a minor revision in 2019, which added several new features.

The purpose of the ISO/IEC 18661 series (first published 2014 through 2016) has been to specify C language support for the new features introduced into IEEE 754 since 1985. Most of the ISO/IEC 18661 series has been incorporated into ISO/IEC 9899:2024 (also referred to as C23 because major work on this revision was completed in 2023), which supports all required and most recommended features in IEEE 754-2019.

IEEE 754 defines alternatives for certain attributes of floating-point semantics, and aims to provide, through programming languages, a 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 constant (fixed at translation time), not dynamic (changeable at execution time).

The `FENV_ROUND` and `FENV_DEC_ROUND` pragmas in C23 provide the rounding direction attributes required by IEEE 754.

IEEE 754 also recommends other attributes that are not supported in C23, including:

— preferredWidth: evaluation formats for floating-point operations;

— value-changing optimizations: allow/disallow program transformations that can affect floating-point result values;

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

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

To supplement the IEEE 754 support in C23, this document provides these recommended attributes by means of standard pragmas. The pragma parameters represent the alternative semantics. The pragmas are similar in form to the floating-point pragmas (`FENV_ACCESS`, `FP_CONTRACT`, `CX_LIMITED_RANGE`) that have been in C since 1999.

STANDARDSISO.COM : Click to view the full PDF of ISO/IEC TS 18661-5:2025

STANDARDSISO.COM : Click to view the full PDF of ISO/IEC TS 18661-5:2025

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

Part 5:
Supplementary attributes

1 Scope

This document specifies extensions to programming language C to include pragmas corresponding to attributes specified and recommended in ISO/IEC 60559 but not supported in ISO/IEC 9899:2024 (also referred to as C23).

2 Normative references

The following documents, in whole or in part, are normatively referenced in this document and are indispensable for its application. For dated references, only the edition cited applies. For undated references, the latest edition of the referenced document (including any amendments) applies.

ISO/IEC 9899:2024, *Information technology — Programming languages — C*

ISO/IEC 60559:2020, *Information technology — Microprocessor Systems — Floating-Point arithmetic*

3 Terms and definitions

For the purposes of this document, the terms and definitions given in ISO/IEC 9899:2024 and ISO/IEC 60559:2020 apply.

ISO and IEC maintain terminological databases for use in standardization at the following addresses:

— ISO Online browsing platform: available at https://www.iso.org/obp

— IEC Electropedia: available at https://www.electropedia.org/

4 Conformance

An implementation that meets the requirements for a conforming implementation of C23 may conform to any or all of the four feature sets in this document. The implementation conforms to the feature sets if

a) it defines `__STDC_IEC_60559_BFP__` or `__STDC_IEC_60559_DFP__` or both, indicating support for ISO/IEC 60559 binary or decimal floating-point arithmetic, as specified in C23, Annex F;

and one or more of the following are true:

b) it defines `__STDC_IEC_60559_ATTRIB_EVALUATION_FORMAT__` to `202401L` and provides the features for evaluation formats as specified in this document (Clause 7);

c) it defines `__STDC_IEC_60559_ATTRIB_OPTIMIZATION__` to `202401L` and provides the features for optimization as specified in this document (Clause 8);

d) it defines `__STDC_IEC_60559_ATTRIB_REPRODUCIBLE__` to `202401L` and provides the features for reproducibility as specified in this document (Clause 9);

STANDARDSISO.COM : Click to view the full PDF of ISO/IEC TS 18661-5:2025

e) it defines `__STDC_IEC_60559_ATTRIB_ALTERNATE_EXCEPTION_HANDLING__` to `202401L` and provides the features for alternate exception handling as specified in this document (Clause 10).

5 C standard conformance

5.1 Freestanding implementations

C23, Clause 4 allows freestanding implementations to conform to this document.

5.2 Predefined macros

The implementation defines one or more of the following macros to indicate conformance to the specification in this document for support of the corresponding attributes specified and recommended in ISO/IEC 60559.

`__STDC_IEC_60559_ATTRIB_EVALUATION_FORMAT__` The integer constant `202401L`.

`__STDC_IEC_60559_ATTRIB_OPTIMIZATION__` The integer constant `202401L`.

`__STDC_IEC_60559_ATTRIB_REPRODUCIBLE__` The integer constant `202401L`.

`__STDC_IEC_60559_ATTRIB_ALTERNATE_EXCEPTION_HANDLING__` The integer constant `202401L`.

5.3 Standard headers

The identifiers specified in this document are defined or declared by the associated header if and only if the implementation defines the relevant feature macros (5.2) and

```
__STDC_WANT_IEC_60559_ATTRIB_EXT__
```

is defined as a macro at the point in the source file where the header is first included.

6 Standard pragmas

C23 provides standard pragmas (C23, 6.10.8) for specifying certain attributes pertaining to floating-point behavior within a compound statement or file. This document extends this practice by introducing additional standard pragmas to support attributes recommended by ISO/IEC 60559:

```
#pragma STDC FP_FLT_EVAL_METHOD width
#pragma STDC FP_DEC_EVAL_METHOD width
#pragma STDC FP_ALLOW_VALUE_CHANGING_OPTIMIZATION on-off-switch
#pragma STDC FP_ALLOW_ASSOCIATIVE_LAW on-off-switch
#pragma STDC FP_ALLOW_DISTRIBUTIVE_LAW on-off-switch
#pragma STDC FP_ALLOW_MULTIPLY_BY_RECIPROCAL on-off-switch
#pragma STDC FP_ALLOW_ZERO_SUBNORMAL on-off-switch
#pragma STDC FP_ALLOW_CONTRACT_FMA on-off-switch
#pragma STDC FP_ALLOW_CONTRACT_OPERATION_CONVERSION on-off-switch
#pragma STDC FP_ALLOW_CONTRACT on-off-switch
#pragma STDC FP_REPRODUCIBLE on-off-switch
#pragma STDC FENV_EXCEPT action except-list
```

width: specified with the pragmas (7.2, 7.3)

on-off-switch: specified in C23, 6.10.8

action, *except-list*: specified with the pragma (10.1)

STANDARDSISO.COM : Click to view the full PDF of ISO/IEC TS 18661-5:2025

7 Evaluation formats

7.1 General

This clause applies to implementations that define:

```
__STDC_IEC_60559_ATTRIB_EVALUATION_FORMAT__
```

C23 gives implementations the flexibility to evaluate operations to the format of the wider operand or to a still wider evaluation format. The values of the macros `FLT_EVAL_METHOD` (C23, 5.3.5.3.3 and H.3) and `DEC_EVAL_METHOD` (C23, 5.3.5.3.4 and H.3) characterize these evaluation methods. Though C23 does not provide means for the user to control the evaluation method, some implementations provide such controls as extensions. ISO/IEC 60559 recommends an attribute for this purpose. The following subclauses (7.2 and 7.3) specify pragmas in `<math.h>` to control the evaluation method. These evaluation method pragmas, like the `FENV_ROUND` (C23, 7.6.3) and `FENV_DEC_ROUND` (C23, 7.6.4) pragmas, affect translation-time expression evaluation and constants.

NOTE As specified in C23, 6.7.2, the value of the initializer for an object declared with storage-class specifier `constexpr` is constrained to be exactly representable in the target type. Thus, an evaluation method pragma whose scope includes the declaration can affect the validity of the declaration.

7.2 Evaluation method pragma

Synopsis

```
#define __STDC_WANT_IEC_60559_ATTRIB_EXT__
#include <math.h>
#pragma STDC FP_FLT_EVAL_METHOD width
```

Constraints

The *width* parameter shall be `-1`, `0`, `DEFAULT`, or another value supported by the implementation.

Description

The `FP_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*. The parameter *width* is an expression in one of the forms:

`0`
decimal-constant
– *decimal-constant*
`DEFAULT`

where the value of the expression is a possible value of the `FLT_EVAL_METHOD` macro, as specified in C23, 5.3.5.3.3 and H.3. An expression represents the evaluation method corresponding to its value (C23, 5.3.5.3.3 and H.3) and `DEFAULT` designates the implementation's default evaluation method (characterized by the `FLT_EVAL_METHOD` macro). The *width* parameter may be `-1`, `0`, or `DEFAULT`. Which, if any, other values of *width* are supported is implementation-defined. 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 `FP_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 occurrence until another `FP_FLT_EVAL_METHOD` 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.

STANDARDSISO.COM : Click to view the full PDF of ISO/IEC TS 18661-5:2025

7.3 Evaluation method pragma for decimal floating types

Synopsis

```
#define __STDC_WANT_IEC_60559_ATTRIB_EXT__
#include <math.h>
#pragma STDC FP_DEC_EVAL_METHOD width
```

Constraints

The *width* parameter shall be `-1`, `1`, `DEFAULT`, or another value supported by the implementation.

Description

The `FP_DEC_EVAL_METHOD` pragma sets the evaluation method for decimal interchange and extended floating types to the evaluation method represented by *width*. The parameter *width* is an expression in one of the forms:

`0`
decimal-constant
– *decimal-constant*
`DEFAULT`

where the value of the expression is a possible value of the `DEC_EVAL_METHOD` macro, as specified in C23, 5.3.5.3.4 and H.3. An expression represents the evaluation method corresponding to its value (C23, 5.3.5.3.4 and H.3) and `DEFAULT` designates the implementation's default evaluation method (characterized by the `DEC_EVAL_METHOD` macro). The *width* parameter may be `-1`, `1`, or `DEFAULT`. Which, if any, other values of *width* are supported is implementation-defined. 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 `FP_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 occurrence until another `FP_DEC_EVAL_METHOD` 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.

7.4 Effective evaluation method macros

The `<float.h>` macros `FLT_EVAL_METHOD` (C23, 5.3.5.3.3 and H.3) and `DEC_EVAL_METHOD` (C23, 5.3.5.3.4 and H.3) characterize the default evaluation method. Their values are constant expressions, suitable for use in conditional expression inclusion preprocessing directives. They are not affected by the evaluation method pragmas, so it is possible they do not reflect the effective evaluation method.

The `<math.h>` header defines macros `FLT_EVAL_METHOD_EFFECTIVE` and `DEC_EVAL_METHOD_EFFECTIVE` that are similar to the `<float.h>` macros `FLT_EVAL_METHOD` and `DEC_EVAL_METHOD`, except that they characterize the effective evaluation method at the point in the program where the macro is used. Thus, they reflect the state of any evaluation method pragmas (7.2, 7.3) that are in effect. These macros shall not be used in conditional expression inclusion preprocessing directives.

7.5 Evaluation type macros

The `<math.h>` types with an `_t` suffix (e.g. `float_t)` (C23, 7.12.1 and H.11), which are defined to match evaluation formats, reflect the evaluation method where no evaluation method pragma is in effect. For each of these types, there is a type-like macro in `<math.h>` with the same name which expands to a designation for the type whose range and precision are used for evaluating operations and constants of the corresponding standard, binary, or decimal floating type. The macro reflects the actual evaluation method, which can be determined by an evaluation method pragma. Use of `#undef` to remove the macro definition will ensure that the actual type is referred to (as though no evaluation method pragma was in effect).

STANDARDSISO.COM : Click to view the full PDF of ISO/IEC TS 18661-5:2025

7.6 Evaluation formats for `<tgmath.h>`

The evaluation methods in C23 apply to floating-point operators, but not to math functions. Hence, they do not apply to the ISO/IEC 60559 operations that are provided as library functions. This subclause specifies a macro the user can define to cause the generic macros in `<tgmath.h>` to be evaluated like floating-point operators.

Except for functions that round result to a narrower type, if the macro

```
__STDC_TGMATH_OPERATOR_EVALUATION__
```

is defined at the point in the program where `<tgmath.h>` is first included, the format of the generic parameters of the function invoked by a type-generic macro is the evaluation format determined by the effective evaluation method (see C23, 5.3.5.3.3, 5.3.5.3.4 and H.3) applied to the types of the arguments for generic parameters. The semantic type of the expanded type-generic macro is as determined by the rules in C23, 7.27 and H.13 and is unchanged by the evaluation method. Neither the arguments for generic parameters nor the result are narrowed to their semantic types. Thus, (if the macro `__STDC_TGMATH_OPERATOR_EVALUATION__` is appropriately defined) the evaluation method affects the operations provided by type-generic macros and floating-point operators in the same way. See 7.6, EXAMPLE.

The macro `__STDC_TGMATH_OPERATOR_EVALUATION__` does not alter the conversion of classification macro arguments to their semantic types (as specified in C23, 7.12.4).

EXAMPLE The following code uses wide evaluation to avoid overflow and underflow.

```
#define __STDC_WANT_IEC_60559_ATTRIB_EXT__
#define __STDC_TGMATH_OPERATOR_EVALUATION__
#include <tgmath.h>
{
    #pragma STDC FLT_EVAL_METHOD 1 /* to double */
    float x, y, z;
    ...
    z = sqrt(x * x + y * y);
}
```

Because of the evaluation method pragma, the sum of squares, whose semantic type is `float`, is evaluated with the range and precision of `double`, hence does not overflow or underflow. The expanded `<tgmath.h>` macro `sqrt` acquires the semantic type of its argument: `float`. However, because the macro `__STDC_TGMATH_OPERATOR_EVALUATION__` is defined before the inclusion of `<tgmath.h>`, the `sqrt` macro behaves like an operator with respect to the evaluation method and does not narrow its argument to its semantic type. Without the definition of the macro `__STDC_TGMATH_OPERATOR_EVALUATION__`, the `sqrt` macro would expand to `sqrtf`, and its evaluated argument would be converted to `float`, which can overflow or underflow.

8 Optimization controls

8.1 General

This clause applies to implementations that define:

```
__STDC_IEC_60559_ATTRIB_OPTIMIZATION__
```

ISO/IEC 60559 recommends attributes to allow and disallow value-changing optimizations, individually and collectively. C23, Annex F disallows value-changing optimizations, except for contractions (which can be controlled as a group with the `FP_CONTRACT` pragma). This clause provides pragmas to allow or disallow certain value-changing optimizations, including those mentioned in ISO/IEC 60559.

The pragmas in this clause can be used to allow the implementation to do certain floating-point optimizations that are generally disallowed because the optimization can change values of floating-point expressions. These pragmas apply to all floating types. It is unspecified whether optimizations allowed by these pragmas occur consistently, or at all. These pragmas (among other standard pragmas) apply to user code. They do not apply to code for operators or library functions that is placed inline by the implementation.

STANDARDSISO.COM : Click to view the full PDF of ISO/IEC TS 18661-5:2025

Some of the pragmas allow optimizations based on identities of real number arithmetic that are not valid for floating-point arithmetic (C23, 5.2.2.4 and F.9.3). Optimizations based on identities that are valid for the implementation's floating-point arithmetic are always allowed. Optimizations based on identities derived from identities whose use is allowed (either by a standard pragma or by virtue of being valid for the implementation's floating-point arithmetic) may also be done.

These pragmas do not affect the requirements on volatile or atomic variables.

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; at the end of a compound statement the state for allowing each optimization controlled by the pragma is restored to its condition just before the compound statement.

8.2 The `FP_ALLOW_VALUE_CHANGING_OPTIMIZATION` pragma

Synopsis

```
#define __STDC_WANT_IEC_60559_ATTRIB_EXT__
#include <math.h>
#pragma STDC FP_ALLOW_VALUE_CHANGING_OPTIMIZATION on-off-switch
```

Description

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

NOTE The `FP_ALLOW_VALUE_CHANGING_OPTIMIZATION` pragma does not affect the evaluation methods. Nevertheless, an evaluation method characterized by a negative value of *width* (C23, 5.3.5.3.3, 5.3.5.3.4 and H.3) can allow for indeterminable evaluation formats, hence unspecified result values.

8.3 The `FP_ALLOW_ASSOCIATIVE_LAW` pragma

Synopsis

```
#define __STDC_WANT_IEC_60559_ATTRIB_EXT__
#include <math.h>
#pragma STDC FP_ALLOW_ASSOCIATIVE_LAW on-off-switch
```

Description

This pragma allows or disallows optimizations 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"

It should be noted that this pragma allows optimizations based on similar mathematical identities involving subtraction and division. For example, for ISO/IEC 60559 floating-point arithmetic, since the identity

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

STANDARDSISO.COM : Click to view the full PDF of ISO/IEC TS 18661-5:2025

is valid (C23, F.9.2), this pragma also allows optimizations based on

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

Similarly, if the states for this pragma and the `FP_ALLOW_MULTIPLY_BY_RECIPROCAL` pragma (8.5) are both "on", then optimizations based on the following are allowed:

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

It should also be noted that for ISO/IEC 60559 floating-point arithmetic, since the commutative laws

$x + y = y + x$

$x \times y = y \times x$

are valid, the pragma allows optimizations based on identities derived from the associative and commutative laws, such as

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

8.4 The `FP_ALLOW_DISTRIBUTIVE_LAW` pragma

Synopsis

```
#define __STDC_WANT_IEC_60559_ATTRIB_EXT__
#include <math.h>
#pragma STDC FP_ALLOW_DISTRIBUTIVE_LAW on-off-switch
```

Description

This pragma allows or disallows optimizations based on the distributive laws for multiplication and division

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

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

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

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

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"

8.5 The `FP_ALLOW_MULTIPLY_BY_RECIPROCAL` pragma

Synopsis

```
#define __STDC_WANT_IEC_60559_ATTRIB_EXT__
#include <math.h>
#pragma STDC FP_ALLOW_MULTIPLY_BY_RECIPROCAL on-off-switch
```

Description

This pragma allows or disallows optimizations based on the mathematical equivalence of division and multiplication by the reciprocal of the denominator

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

STANDARDSISO.COM : Click to view the full PDF of ISO/IEC TS 18661-5:2025

where *on-off-switch* is one of

`ON` – allow multiply by reciprocal

`OFF` – do not allow multiply by reciprocal

`DEFAULT` – “off”

8.6 The `FP_ALLOW_ZERO_SUBNORMAL` pragma

Synopsis

```
#define __STDC_WANT_IEC_60559_ATTRIB_EXT__
#include <math.h>
#pragma STDC FP_ALLOW_ZERO_SUBNORMAL on-off-switch
```

Description

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”

Within the scope of this pragma, the floating-point operations affected by the pragma are all floating-point operators, implicit conversions (including the conversion of a value represented in a format wider than its semantic type to its semantic type, as done by classification macros), and invocations of applicable functions in `<math.h>`, `<stdio.h>`, `<stdlib.h>`, and `<wchar.h>` for which macro replacement has not been suppressed (C23, 7.1.4). Thus, subnormal operands and results of affected operations may be replaced by zero. Whether the replacement raises the “inexact” and “underflow” floating-point exceptions is unspecified. Functions not affected by the pragma behave as though no `FP_ALLOW_ZERO_SUBNORMAL` pragma were in effect at the site of the call.

8.7 The `FP_ALLOW_CONTRACT_FMA` pragma

Synopsis

```
#define __STDC_WANT_IEC_60559_ATTRIB_EXT__
#include <math.h>
#pragma STDC FP_ALLOW_CONTRACT_FMA on-off-switch
```

Description

This pragma allows or disallows contraction (C23, 6.5.1) of floating-point multiply and add or subtract (with the result of the multiply)

$x * y + z$

$x * y - z$

$x + y * z$

$x - y * z$

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

STANDARDSISO.COM : Click to view the full PDF of ISO/IEC TS 18661-5:2025

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

NOTE ISO/IEC 60559 uses the term “synthesize” instead of “contract”.

8.8 The `FP_ALLOW_CONTRACT_OPERATION_CONVERSION` pragma

Synopsis

```
#define __STDC_WANT_IEC_60559_ATTRIB_EXT__
#include <math.h>
#pragma STDC FP_ALLOW_CONTRACT_OPERATION_CONVERSION on-off-switch
```

Description

This pragma allows or disallows contraction (C23, 6.5.1) 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”

Within the scope of this pragma, the floating-point operations affected by the pragma are all floating-point operators, implicit conversions (including the conversion of a value represented in a format wider than its semantic type to its semantic type, as done by classification macros), and invocations of applicable functions in `<math.h>`, `<stdio.h>`, `<stdlib.h>`, and `<wchar.h>` for which macro replacement has not been suppressed (C23, 7.1.4). Thus, an affected operation may be contracted with a conversion of its result. Functions not affected by the pragma behave as though no `FP_ALLOW_CONTRACT_OPERATION_CONVERSION` pragma were in effect at the site of the call.

EXAMPLE For the code sequence

```
#define __STDC_WANT_IEC_60559_ATTRIB_EXT__
#include <math.h>
#pragma STDC FP_ALLOW_CONTRACT_OPERATION_CONVERSION ON
float f1, f2;
double d1, d2;
…
f1 = d1 * d2;
f2 = sqrt(d1);
```

the multiply (operation) and assignment (conversion) can 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 be rounded according to the evaluation method and the assignment would entail a second rounding. With the given code, the `sqrt` function can be replaced by `fsqrt`, avoiding the need for a separate operation to convert the `double` result of `sqrt` to `float`.

8.9 The `FP_ALLOW_CONTRACT` pragma

Synopsis

```
#define __STDC_WANT_IEC_60559_ATTRIB_EXT__
#include <math.h>
#pragma STDC FP_ALLOW_CONTRACT on-off-switch
```

Description

This pragma allows or disallows contraction (C23, 6.5.1) for floating-point operations, where *on-off-switch* is one of

`ON` – allow contraction for floating-point operations

STANDARDSISO.COM : Click to view the full PDF of ISO/IEC TS 18661-5:2025

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

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

The optimizations controlled by this pragma include those controlled by the `FP_ALLOW_CONTRACT_FMA` and `FP_ALLOW_CONTRACT_OPERATION_CONVERSION` pragmas.

This pragma is equivalent to the `FP_CONTRACT` pragma (C23, 7.12.3), also in `<math.h>`: the two pragmas may be used interchangeably, provided the implementation defines `__STDC_WANT_IEC_60559_ATTRIB_EXT__`.

9 Reproducibility

9.1 General

This clause applies to implementations that define

`__STDC_IEC_60559_ATTRIB_REPRODUCIBLE__`

ISO/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 ISO/IEC 60559, so that no unspecified or implementation-defined behavior is admitted.

This clause provides a pragma to support the ISO/IEC 60559 attribute for reproducible results and gives requirements for programs to have reproducible results. Where the state of the pragma is “on”, floating-point numerical results and exception flags are reproducible (given the same inputs, including relevant environment variables) on implementations that define

`__STDC_IEC_60559_ATTRIB_REPRODUCIBLE__`

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 (9.3).

An implementation that defines `__STDC_IEC_60559_ATTRIB_REPRODUCIBLE__` 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 code using (binary) types `float` and `double`. If the implementation defines `__STDC_IEC_60559_DFP__`, it supports reproducible results for code using types `_Decimal32`, `_Decimal64`, and `_Decimal128`. If the implementation defines `__STDC_IEC_60559_TYPES__`, then it supports reproducible results for code using its interchange floating types (C23, H.2.2). If the implementation provides a set of correctly rounded math functions (C23, 7.33.9), then it supports reproducible results for code using correctly rounded math functions from that set.

9.2 The `FP_REPRODUCIBLE` pragma

Synopsis

```
#define __STDC_WANT_IEC_60559_ATTRIB_EXT__
#include <math.h>
#pragma STDC FP_REPRODUCIBLE on-off-switch
```

STANDARDSISO.COM : Click to view the full PDF of ISO/IEC TS 18661-5:2025

Description

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 `FP_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 `FP_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.

If the state of the pragma is "on", then the effects of the following are implied:

```
#pragma STDC FENV_ACCESS ON
#pragma STDC FP_ALLOW_VALUE_CHANGING_OPTIMIZATION OFF
```

and if `__STDC_IEC_60559_BFP__` is defined

```
#pragma STDC FP_FLT_EVAL_METHOD 0
```

and if `__STDC_IEC_60559_DFP__` is defined

```
#pragma STDC FP_DEC_EVAL_METHOD 1
```

If the `FP_REPRODUCIBLE` pragma appears with the *on-off-switch* `OFF` under the effect of a `FP_REPRODUCIBLE` 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 `FP_REPRODUCIBLE` pragma with on-off-switch `ON`. The `FP_REPRODUCIBLE` pragma with *on-off-switch* `OFF` has no effect if it occurs where the state of the pragma is "off".

The default state of the pragma is "off".

Recommended practice: The implementation is encouraged to issue a diagnostic message if, where the state of the `FP_REPRODUCIBLE` pragma is "on", the source code uses a language or library feature whose results may not be reproducible.

9.3 Reproducible code

The following properties support code sequences in producing reproducible results.[1)]

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

— All floating-point operations used by the code are bound to ISO/IEC 60559 operations, as described in C23, F.3 in Table F.2.

— The code does not contain any use that may result in undefined behavior. The code does not depend on any behavior that is unspecified, implementation-defined, or locale-specific.

The restrictive properties below are examples, not a complete list. See also C23, Annex J. Although the properties may not be necessary in all cases for reproducible code, the user is advised to follow the restrictions to avoid common programming practices that would undermine reproducibility.

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

— The code does not use complex or imaginary types.

1) Of course, if the code uses optional features, results will be reproducible only on implementations that support those features.

STANDARDSISO.COM : Click to view the full PDF of ISO/IEC TS 18661-5:2025

— If `__STDC_IEC_60559_BFP__` is not defined by the implementation, the code does not use the `float` or `double` 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 depend on the payloads (C23, F.10.14) or sign bits of quiet NaNs.

— The code does not use signaling NaNs.

— 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 ISO/IEC 60559 which specifies that an implementation supporting reproducibility shall 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.

— The code does not depend on the quantum of a result for the decimal maximum and minimum functions in C23, 7.12.13 when the arguments are equal.

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

— 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 `FP_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 `FP_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 use an `FENV_EXCEPT` pragma (10.2) with an *action* `OPTIONAL_FLAG`, `BREAK`, `TRY`, or `CATCH`.

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

10 Alternate exception handling

10.1 General

This clause applies to implementations that define:

```
__STDC_IEC_60559_ATTRIB_ALTERNATE_EXCEPTION_HANDLING__
```

ISO/IEC 60559 arithmetic raises floating-point exceptions to inform the program when an operation encounters problematic inputs, such that no one result would be suitable for all situations. The default exception handling in ISO/IEC 60559 is intended to be more useful in more situations than other schemes, or at least predictable. However, other exception handling is more useful in certain situations. Thus, ISO/IEC 60559 describes alternate exception handling and recommends that programming languages provide means for the program to specify which exception handling will be done.

When a floating-point exception is raised, the ISO/IEC 60559 default exception handling sets the appropriate exception flag(s), returns a specified result, and continues execution. ISO/IEC 60559 also prescribes alternate exception handling. The pragma in this clause provides a means for the program to choose the method of exception handling. The pragma applies to operations on all floating types.

STANDARDSISO.COM : Click to view the full PDF of ISO/IEC TS 18661-5:2025

For the “underflow” exception, the chosen exception handling occurs if the exception is raised, whether the default result would be exact or inexact, unless stated otherwise.

10.2 The `FENV_EXCEPT` pragma

Synopsis

```
#define __STDC_WANT_IEC_60559_ATTRIB_EXT__
#include <fenv.h>
#pragma STDC FENV_EXCEPT action except-list
```

Description

The `FENV_EXCEPT` pragma sets the method specified by *action* for handling the exceptions represented by *except-list*.

except-list shall be a comma-separated list of distinct supported exception designations (or one supported exception designation). The supported exception designations shall include the exception macro identifiers (C23, 7.6.1)

```
FE_DIVBYZERO
FE_INEXACT
FE_INVALID
FE_OVERFLOW
FE_UNDERFLOW
FE_ALL_EXCEPT
```

The `<fenv.h>` header should define macros for the following sub-exceptions and may define additional macros with the appropriate prefix (`FE_INVALID_` or `FE_DIVBYZERO_`) for other sub-exceptions. The supported exception designations shall include the defined sub-exception macro identifiers (if any). If defined, the macros expand to integer constant expressions. Sub-exceptions corresponding to defined macros occur as specified below, and not in other cases.

— “invalid” floating-point exceptions from add and subtract operators and functions that add or subtract (C23, 7.12.15.2, 7.12.15.3 and F.10.12), not caused by signaling NaN input

```
FE_INVALID_ADD
```

— “invalid” floating-point exceptions from divide operators and functions that divide (C23, 7.12.15.5 and F.10.12), not caused by signaling NaN input

```
FE_INVALID_DIV
```

— “invalid” floating-point exceptions from functions that compute multiply-add (C23, 7.12.14.1, F.10.11.1, 7.12.15.6 and F.10.12) and from contracted multiply and add operators, not caused by signaling NaN input

```
FE_INVALID_FMA
```

— “invalid” floating-point exceptions from conversions from floating to integer types (C23, F.4), not caused by signaling NaN input

```
FE_INVALID_INT
```

— “invalid” floating-point exceptions from `ilogb` and `llogb` functions (C23, F.10.4.8 and F.10.4.10), not caused by signaling NaN input

```
FE_INVALID_ILOGB
```

— “invalid” floating-point exceptions from multiply operators and functions that multiply (C23, 7.12.15.4 and F.10.12), not caused by signaling NaN input

```
FE_INVALID_MUL
```

STANDARDSISO.COM : Click to view the full PDF of ISO/IEC TS 18661-5:2025

— “invalid” floating-point exceptions from the `quantized`*N* functions (C23, 7.12.16.1), not caused by signaling NaN input

    FE_INVALID_QUANTIZE

— “invalid” floating-point exceptions from the `remainder` and `remquo` functions (C23, F.10.8.2 and F.10.8.3), not caused by signaling NaN input

    FE_INVALID_REM

— “invalid” floating-point exceptions from functions that compute square root or reciprocal of square root (C23, F.10.5.9, F.10.5.10, 7.12.15.7 and F.10.12), not caused by signaling NaN input

    FE_INVALID_SQRT

— “invalid” floating-point exceptions caused by signaling NaN input (C23, F.2.2)

    FE_INVALID_SNAN

— “invalid” floating-point exceptions from relational operators and comparison macros (C23, 6.5.9, 7.12.18 and F.10.15.2), not caused by signaling NaN input

    FE_INVALID_UNORDERED

— “divide-by-zero” floating-point exceptions from divide operators and functions that divide (C23, 7.12.15.5 and F.10.12)

    FE_DIVBYZERO_ZERO

— “divide-by-zero” floating-point exceptions from logarithm and `logb` functions (C23, F.10.4.11, F.10.4.12, F.10.4.13, F.10.4.14, F.10.4.15, F.10.4.16 and F.10.4.17)

    FE_DIVBYZERO_LOG

action shall be a designation of a supported exception handling method. The following actions shall be provided:

— default exception handling (as specified in ISO/IEC 60559).

    DEFAULT

— default exception handling, but without setting the flag.

    NO_FLAG

— default exception handling, but whether the flag is set (as with default exception handling), and for which operations and their occurrences, is unspecified.

    OPTIONAL_FLAG

— *abrupt underflow*. If an “underflow” floating-point exception occurs (see ISO/IEC 60559), the operation delivers a result with magnitude zero or the minimum normal magnitude (for the result format) and with the same sign as the default result, sets the “underflow” floating-point exception flag, and raises the “inexact” floating-point exception. When

rounding to nearest, ties to even

rounding to nearest, ties away from zero

or

rounding toward zero

the result magnitude is zero. When rounding toward positive infinity, the result magnitude is the minimum normal magnitude if the result sign is positive, and zero if the result sign is negative. When

STANDARDSISO.COM : Click to view the full PDF of ISO/IEC TS 18661-5:2025

rounding toward negative infinity, the result magnitude is the minimum normal magnitude if the result sign is negative, and zero if the result sign is positive. Abrupt underflow has no effect on the interpretation of subnormal operands. The *action* has no effect if `FE_UNDERFLOW` is not included in *except-list*.

```
ABRUPT_UNDERFLOW
```

With one of the actions in the list above, the pragma shall occur either outside external declarations, or preceding all explicit declarations and statements inside a compound statement, which then is the compound statement associated with the pragma. When outside external declarations, the pragma action for a designated exception takes effect from the occurrence of the pragma until another `FENV_EXCEPT` pragma designating the same exception is encountered, or until the end of the translation unit. When inside a compound statement, the pragma action for a designated exception takes effect from the occurrence of the pragma until another `FENV_EXCEPT` pragma designating the same exception 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 handling each exception designated in *except-list* is restored to its condition just before the compound statement.

The actions in the lists below in this subclause affect flow of control. With one of these actions, the pragma shall precede a compound statement, which is the compound statement associated with the pragma. There shall be nothing between the pragma and its associated compound statement except perhaps white space (including comments). For a designated exception, the pragma takes effect from the beginning of the associated compound statement until another `FENV_EXCEPT` pragma designating the same exception is encountered (with a nested associated compound statement), or until the end of the compound statement. At the end of a compound statement the state for handling each exception designated in *except-list* is restored to its condition just before the compound statement.

— *break*. Terminate execution of the compound statement associated with the pragma. Then, continue execution after the associated compound statement. When termination occurs, the following apply: if the execution to completion of the associated compound statement (without the break) would at any point modify an object, the value of the object is indeterminate; if the execution would modify the state of the dynamic rounding mode or any state maintained by the standard library (e.g. in the I/O system), the state is unspecified; the values of flags for the designated exceptions are unspecified. (Thus, termination may occur as soon as possible after the exception is raised, to maximize performance.)

```
BREAK
```

The following two actions work together. A compound statement associated with a *try* action shall be paired with one or more compound statements each associated with a *catch* action. The pragmas with *catch* actions and their associated compound statements shall appear contiguously immediately below the compound statement associated with the *try* action, except for white space (including comments). Each exception designation in the pragma with a *try* action shall appear in one and only one of the pragmas with a *catch* action.

— *try*. The designated exceptions may be handled by a *catch* action. It is unspecified whether flags for designated exceptions that are set in the execution of the associated compound statement are restored to their states before the associated compound statement. The associated compound statement shall not be the *statement* of a selection (C23, 6.8.4) or iteration (C23, 6.8.5) statement. There shall be no jumps into or out of the associated compound statement, other than to handle an exception, as specified below.

```
TRY
```

— *catch*. If any designated exception occurs in the execution of the compound statement associated with the *try* action, jump to a compound statement associated with some *catch* action with an occurring designated exception. Upon completion of the associated compound statement, continue execution after the last of the compound statements associated with *catch* actions. The jump target should be a compound statement associated with the first occurring designated exception. When the jump occurs, the following apply: if the execution of the associated compound statement to completion (without the jump) would at any point modify an object, the value of the object is indeterminate; if the execution would modify the state of the dynamic rounding mode or any state maintained by the standard library (e.g., in the I/O system), the state is unspecified. (Thus, the jump may occur as soon as possible after the exception is raised, to maximize performance). The compound statement associated with a *catch* action is executed

STANDARDSISO.COM : Click to view the full PDF of ISO/IEC TS 18661-5:2025

only to handle an exception occurring in the compound statement associated with the *try* action. There shall be no other jumps into or out of the compound statement associated with a *catch* action.

```
CATCH
```

NOTE The compound statements associated with a *try* action and its *catch* actions (or with a *delayed-try* action and its *delayed-catch* actions), together enclosed in braces, may be the *statement* of a selection or iteration statement. For example, the following code segment is permitted:

```
for (int i = 0; i < LEN; i++) {
    #pragma STDC FENV_EXCEPT TRY FE_OVERFLOW
    {
        y[i] = x[i] * x[i];
    }
    #pragma STDC FENV_EXCEPT CATCH FE_OVERFLOW
    {
        y[i] = DBL_MAX;
    }
}
```

The following two actions work together. A compound statement associated with a *delayed-try* action shall be paired with one or more compound statements each associated with a *delayed-catch* action. The pragmas with *delayed-catch* actions and their associated compound statements shall appear contiguously immediately below the compound statement associated with the *delayed-try* action, except for white space (including comments). Each exception designation in the pragma with a *delayed-try* action shall appear in one and only one of the pragmas with a *delayed-catch* action. For supported sub-exceptions, the behavior of the actions listed below shall be as if the exceptions, flags, and functions in the specification were extended for sub-exceptions, though such extensions are not prescribed in this document.

— *delayed-try*. The designated exceptions may be handled by a *delayed-catch* action. Before executing the compound statement associated with the *delayed-try* action, save (as by `fegetexceptflag`) the states of the flags for the designated exceptions, and then clear (as by `feclearexcept`) the designated exceptions. After normal completion of the associated compound statement, re-save the states of the designated exceptions. Then restore (as by `fesetexceptflag`) the designated exception flag states before the associated compound statement. The associated compound statement shall not be the *statement* of a selection (C23, 6.8.5) or iteration (C23, 6.8.6) statement. There shall be no jumps into or out of the associated compound statement.

```
DELAYED_TRY
```

— *delayed-catch*. Test (as by `fetestexceptflag`) the exception flag states saved after completion of the compound statement associated with the *delayed-try* action. If any exception with the same designation for the *delayed-try* action and a *delayed-catch* action occurred (as determined by flag state tests), jump to the first compound statement associated with an occurring exception with the same designation for the *delayed-try* action and a *delayed-catch* action. Upon completion of the associated compound statement, continue execution after the last of the compound statements associated with *delayed-catch* actions. Each exception designation shall be listed in at most one of the pragmas with a *delayed-catch* action. The compound statement associated with a *delayed-catch* action is executed only to handle an exception occurring in the compound statement associated with the *delayed-try* action. There shall be no other jumps into or out of the compound statement associated with a *delayed-catch* action.

```
DELAYED_CATCH
```

Within the scope of an `FENV_EXCEPT` pragma, the floating-point operations affected by the pragma are all floating-point operators, implicit conversions (including the conversion of a value represented in a format wider than its semantic type to its semantic type, as done by classification macros), and invocations of applicable functions in `<math.h>`, `<stdio.h>`, `<stdlib.h>`, and `<wchar.h>` for which macro replacement has not been suppressed (C23, 7.1.4). Thus, exceptions raised by affected operations are handled according to the specified *action*. Functions not affected by the pragma behave as though no `FENV_EXCEPT` pragma were in effect at the site of the call.

STANDARDSISO.COM : Click to view the full PDF of ISO/IEC TS 18661-5:2025