Technical Specification

ISO/IEC TS 18661-4

Second edition
2025-03

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

Part 4:
Supplementary functions

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

Partie 4: Fonctions supplémentaires

Reference number
ISO/IEC TS 18661-4:2025(en)

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.

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 18661-4:2015), which has been technically revised.

The main changes are as follows:

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

— The mathematical functions and constant rounding modes have been removed. These features are now incorporated into ISO/IEC 9899:2024.

— Functions to support the augmented arithmetic operations specified in IEEE 754-2019 have been added.

— New headers have been added, and all extensions to the `<math.h>` header have been removed.

A list of all parts in the ISO 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.

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.

To supplement the IEEE 754 support in C23, this document specifies two C headers with functions corresponding to the reduction and augmented arithmetic operations recommended by IEEE 754, but not included in C23.

The reduction operations perform widely used vector computations involving sums and products, including scaled products. These operations are allowed to associate in any order, and to evaluate in any wider format.

The augmented arithmetic operations, added in IEEE 754-2019, are versions of operations commonly called twoSum and twoProduct. These operations can be used to implement arithmetic with extra precision, for example, for double-double format. In theory, they can also be used to implement efficient reproducible dot products.

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

Part 4: Supplementary functions

1 Scope

This document specifies extensions to programming language C to include functions corresponding to operations 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 one or both feature sets (reduction functions and augmented arithmetic) in this document. The implementation conforms to the reduction functions feature 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;

b) it defines `__STDC_IEC_60559_FUNCS_REDUCTION__` to `202401L` and provides the `<reduc.h>` header specified in this document (Clause 6).

The implementation conforms to the augmented arithmetic feature if:

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

d) it defines `__STDC_IEC_60559_FUNCS_AUGMENTED_ARITHMETIC__` to `202401L` and provides the `<augarith.h>` header specified in this document (Clause 7).

5 C standard extensions

5.1 Predefined macros

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

`__STDC_IEC_60559_FUNCS_REDUCTION__` The integer constant `202401L`.

`__STDC_IEC_60559_FUNCS_AUGMENTED_ARITHMETIC__` The integer constant `202401L`.

5.2 Freestanding implementations

The strictly conforming programs that shall be accepted by a conforming freestanding implementation that defines one of the feature macros in 5.1 may also use features in the corresponding header specified in this document. See C23, Clause 4.

5.3 Headers

If the implementation defines one of the feature macros in 5.1 then the implementation provides the corresponding header specified in this document. The header and its use follow the general specification in C23, 7.1 for the C Library as though the header were a subclause of C23, Clause 7 for a conditional feature.

5.4 Future directions

For implementations that define `__STDC_IEC_60559_FUNCS_REDUCTION__`, function names that begin with `reduc_` or `scaled_` are potentially reserved identifiers and may be added to the declarations in the `<reduc.h>` header.

For implementations that define `__STDC_IEC_60559_FUNCS_AUGMENTED_ARITHMETIC__`, tag names that end with `aug_t` and function names that begin with `aug_` are potentially reserved identifiers and may be added to the declarations in the `<augarith.h>` header.

See C23, 7.33.

6 Reduction functions `<reduc.h>`

6.1 General

The header `<reduc.h>` declares the type and functions in this clause.

The type declared is `size_t` (described in C23, 7.21.1).

Each function in this clause is declared in `<reduc.h>` if and only if the corresponding type is supported according to C23, Annex F or Annex H.

The functions in this clause shall be implemented so that intermediate computations do not overflow or underflow. For the `reduc_sum`, `reduc_sumabs`, `reduc_sumsq`, and `reduc_sumprod` functions, the “overflow” or “underflow” floating-point exception is raised and a range error occurs, if and only if the final result overflows or underflows. The `scaled_prod`, `scaled_prodsum`, and `scaled_proddiff` functions do not raise the “overflow” or “underflow” floating-point exceptions and do not cause a range error.

The reduction functions do not raise the “divide-by-zero” floating-point exception.

With ISO/IEC 60559 default exception handling, these functions raise the “inexact” floating-point exception in response to “overflow” and “underflow” exceptions; otherwise, whether they raise the “inexact” floating-point exception is unspecified.

Numerical results and exceptional behavior, including the “invalid” floating-point exception, can differ due to the precision of intermediates and the order of evaluation. However, only one floating-point exception is raised (other than “inexact” in response to “overflow” or “underflow”) per reduction function invocation; exceptions are not raised for each exceptional intermediate operand or result. Reduction functions may raise the “invalid” floating-point exception if an element of an array argument is a signaling NaN (see C23, F.2.2). Once an "invalid" floating-point exception is raised, due to signaling NaN, $\infty-\infty$, or $0\times\infty$, processing of array elements may stop.

Whether and how rounding direction modes affect functions in this clause are implementation defined and may be indeterminate. This applies to constant as well as dynamic rounding modes, C23, 7.6.3 notwithstanding.

The preferred quantum exponent for the reduction functions for decimal floating types is unspecified.

For each of the following synopses, an implementation shall declare the functions suffixed with `f`*N* or `f`*N*`x` only if it supports the corresponding binary floating type and the macro `__STDC_WANT_IEC_60559_TYPES_EXT__` is defined at the point in the code where `<reduc.h>` is first included. An implementation shall declare the functions suffixed with `d`*N* for $N \neq 32$, 64 or 128 or with `d`*N*`x` only if it supports the corresponding decimal floating type and the macro `__STDC_WANT_IEC_60559_TYPES_EXT__` is defined at the point in the code where `<reduc.h>` is first included (see C23, Annex H.)

NOTE For N = 32, 64 and 128, the functions suffixed with `d`*N* are declared if the implementation supports decimal floating types (i.e. defines `__STDC_IEC_60559_DFP__`), without the requirement that the macro `__STDC_WANT_IEC_60559_TYPES_EXT__` be defined.

6.2 The `reduc_sum` functions

Synopsis

```
#include <reduc.h>

#ifdef __STDC_IEC_60559_BFP__
double reduc_sum(size_t n, const double p[static n]);
float reduc_sumf(size_t n, const float p[static n]);
long double reduc_suml(size_t n, const long double p[static n]);
_FloatN reduc_sumfN(size_t n, const _FloatN p[static n]);
_FloatNx reduc_sumfNx(size_t n, const _FloatNx p[static n]);
#endif
#ifdef __STDC_IEC_60559_DFP__
_DecimalN reduc_sumdN(size_t n, const _DecimalN p[static n]);
_DecimalNx reduc_sumdNx(size_t n, const _DecimalNx p[static n]);
#endif
```


Description

The `reduc_sum` functions compute the sum of the `n` elements of array `p`: $\sum_{i=0}^{n-1} p[i]$. If the length `n` = 0, the functions return the value +0. If any element of array `p` is a NaN, the functions return a quiet NaN. If any two elements of array `p` are infinities with different signs, the functions return a quiet NaN and raise the “invalid” floating-point exception and a domain error occurs. Otherwise (if no element of `p` is a NaN and no two elements of `p` are infinities with different signs), if any element of array `p` is an infinity, the functions return that same infinity.

Returns

The `reduc_sum` functions return the computed sum.

6.3 The `reduc_sumabs` functions

Synopsis

```
#include <reduc.h>

#ifdef __STDC_IEC_60559_BFP__
double reduc_sumabs(size_t n, const double p[static n]);
float reduc_sumabsf(size_t n, const float p[static n]);
long double reduc_sumabsl(size_t n, const long double p[static n]);
_FloatN reduc_sumabsfN(size_t n, const _FloatN p[static n]);
_FloatNx reduc_sumabsfNx(size_t n, const _FloatNx p[static n]);
#endif
#ifdef __STDC_IEC_60559_DFP__
_DecimalN reduc_sumabsdN(size_t n, const _DecimalN p[static n]);
_DecimalNx reduc_sumabsdNx(size_t n, const _DecimalNx p[static n]);
#endif
```

Description

The `reduc_sumabs` functions compute the sum of the absolute values of the `n` elements of array `p`: $\sum_{i=0}^{\mathrm{n}-1} |\mathrm{p}[i]|$. If the length `n` = 0, the functions return the value +0. If any element of array `p` is an infinity, the functions return +∞; otherwise, if any element of array `p` is a NaN, the functions return a quiet NaN.

Returns

The `reduc_sumabs` functions return the computed sum.

6.4 The `reduc_sumsq` functions

Synopsis

```
#include <reduc.h>

#ifdef __STDC_IEC_60559_BFP__
double reduc_sumsq(size_t n, const double p[static n]);
float reduc_sumsqf(size_t n, const float p[static n]);
long double reduc_sumsql(size_t n, const long double p[static n]);
_FloatN reduc_sumsqfN(size_t n, const _FloatN p[static n]);
_FloatNx reduc_sumsqfNx(size_t n, const _FloatNx p[static n]);
#endif
#ifdef __STDC_IEC_60559_DFP__
_DecimalN reduc_sumsqdN(size_t n, const _DecimalN p[static n]);
_DecimalNx reduc_sumsqdNx(size_t n, const _DecimalNx p[static n]);
#endif
```

Description

The `reduc_sumsq` functions compute the sum of squares of the values of the `n` elements of array `p`: $\sum_{i=0}^{\mathrm{n}-1} (\mathrm{p}[i] \times \mathrm{p}[i])$. If the length `n` = 0, the functions return the value +0. If any element of array `p` is an infinity, the functions return +∞; otherwise, if any element of array `p` is a NaN, the functions return a quiet NaN.

Returns

The `reduc_sumsq` functions return the computed sum.