TECHNICAL REPORT

ISO/IEC TR 24733

First edition
2011-11-01

Information technology — Programming languages, their environments and system software interfaces — Extensions for the programming language C++ to support decimal floating-point arithmetic

Technologies de l'information — Langages de programmation, leurs environnements et interfaces de logiciel système — Extensions pour le langage de programmation C++ pour supporter l'arithmétique flottante décimale

iTeh STANDARD PREVIEW
(standards.iteh.ai)
ISO/IEC TR 24733:2011
https://standards.iteh.ai/catalog/standards/sist/6b27fd3a-b59c-42da-ab87-10c9fd07faef/iso-iec-tr-24733-2011

Reference number
ISO/IEC TR 24733:2011(E)

© ISO/IEC 2011

iTeh STANDARD PREVIEW
(standards.iteh.ai)
ISO/IEC TR 24733:2011
https://standards.iteh.ai/catalog/standards/sist/6b27fd3a-b59c-42da-ab87-10c9fd07faef/iso-iec-tr-24733-2011

COPYRIGHT PROTECTED DOCUMENT

© ISO/IEC 2011

All rights reserved. Unless otherwise specified, no part of this publication may be reproduced or utilized in any form or by any means, electronic or mechanical, including photocopying and microfilm, without permission in writing from either ISO at the address below or ISO's member body in the country of the requester.

ISO copyright office
Case postale 56 • CH-1211 Geneva 20
Tel. + 41 22 749 01 11
Fax + 41 22 749 09 47
E-mail copyright@iso.org
Web www.iso.org

Published in Switzerland

© ISO/IEC 2011 – All rights reserved

Contents

Page

iTeh STANDARD PREVIEW
(standards.iteh.ai)
ISO/IEC TR 24733:2011
https://standards.iteh.ai/catalog/standards/sist/6b27fd3a-b59c-42da-ab87-10e9fd07beff/iso-iec-tr-24733-2011

© ISO/IEC 2011 – All rights reserved

iTeh STANDARD PREVIEW
(standards.iteh.ai)
ISO/IEC TR 24733:2011
https://standards.iteh.ai/catalog/standards/sist/6b27fd3a-b59c-42da-ab87-10c9fd07faef/iso-iec-tr-24733-2011

© ISO/IEC 2011 – All rights reserved

iTeh STANDARD PREVIEW
(standards.iteh.ai)
ISO/IEC TR 24733:2011
https://standards.iteh.ai/catalog/standards/sist/6b27fd3a-b59c-42da-ab87-10c9fd07faef/iso-iec-tr-24733-2011

© ISO/IEC 2011 – All rights reserved

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.

International Standards are drafted in accordance with the rules given in the ISO/IEC Directives, Part 2.

The main task of the joint technical committee is to prepare International Standards. Draft International Standards adopted by the joint technical committee are circulated to national bodies for voting. Publication as an International Standard requires approval by at least 75 % of the national bodies casting a vote.

In exceptional circumstances, when the joint technical committee has collected data of a different kind from that which is normally published as an International Standard ("state of the art", for example), it may decide to publish a Technical Report. A Technical Report is entirely informative in nature and shall be subject to review every five years in the same manner as an International Standard.

iTeh STANDARD PREVIEW
(standards.iteh.ai)

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.

ISO/IEC TR 24733 was prepared jointly by Joint Technical Committee ISO/IEC JTC 1, *Information technology*, Subcommittee SC 22, *Programming languages, their environments and system software interfaces.*

ISO/IEC TR 24733:2011
10c9fd07faef/iso-iec-tr-24733-2011

© ISO/IEC 2011 – All rights reserved

Introduction

0.1 General

Most of today's general purpose computing architectures provide binary floating-point arithmetic in hardware. Binary float-point is an efficient representation that minimizes memory use, and is simpler to implement than floating-point arithmetic using other bases. It has therefore become the norm for scientific computations, with almost all implementations following the IEEE-754 standard for binary floating-point arithmetic.

However, human computation and communication of numeric values almost always uses decimal arithmetic, and decimal notations. Laboratory notes, scientific papers, legal documents, business reports and financial statements all record numeric values in decimal form. When numeric data are given to a program or are displayed to a user, binary to-and-from decimal conversion is required. There are inherent rounding errors involved in such conversions; decimal fractions cannot, in general, be represented exactly by floating-point values. These errors often cause usability and efficiency problems, depending on the application.

These problems are minor when the application domain accepts, or requires results to have, associated error estimates (as is the case with scientific applications). However, in business and financial applications, computations are required either to be exact (with no rounding errors) unless explicitly rounded, or to be supported by detailed analyses that are auditable to be correct. Such applications therefore have to take special care in handling any rounding errors introduced by the computations.

iTeh STANDARD PREVIEW
(standards.iteh.ai)

The most efficient way to avoid conversion error is to use decimal arithmetic. Recognizing this, the IEEE 754-2008 Standard for Floating-Point Arithmetic specifies decimal floating-point encodings and arithmetic. This technical report specifies extensions to the International Standard for the C++ programming language to permit the use of decimal arithmetic in a manner consistent with the IEEE 754-2008 standard.

ISO/IEC TR 24733:2011
https://standards.iteh.ai/catalog/standards/sist/6b27fd5a-b59c-42da-ab87-0c9fd07faef/iso-iec-tr-24733-2011

0.2 Arithmetic model

This Technical Report is based on a model of decimal arithmetic which is a formalization of the decimal system of numeration (algorism) as further defined and constrained by the relevant standards, IEEE 854, ANSI X3-274, and IEEE 754-2008.

There are three components to the model:

- *data*- numbers and NaNs, which can be manipulated by, or be the results of, the core operations defined in the model
- *operations*- (such as addition, multiplication, etc.) which can be carried out on data
- *context* - the status of operations (namely, exceptions flags), and controls to govern the results of operations (for example, rounding modes).

The model defines these components in the abstract. It defines neither the way in which operations are expressed (which might vary depending on the computer language or other interface being used), nor the concrete representation (specific layout in storage, or in a processor's register, for example) of data or context.

© ISO/IEC 2011 – All rights reserved

From the perspective of the C++ language, *data* are represented by data types, *operations* are defined within expressions, and *context* is the floating environment specified in `<cfenv>` and `<fenv.h>`. This Technical Report specifies how the C++ language implements these components.

0.3 The formats

In the C++ International Standard, the representation of a floating-point number is specified in an abstract form where the constituent components of the representation are defined (sign, exponent, significand) but the internals of these components are not. 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.

This approach was inherited from the C programming language. At the time that C was first standardized, there were already various hardware implementations of floating-point arithmetic in common use. Specifying the exact details of a representation would make most of the existing C implementations at the time not conforming.

This Technical Report specifies decimal floating-point arithmetic according to the IEEE 754-2008 standard, with the constituent components of the representation defined. This is more stringent than the approach taken for the floating-point types in the C++ standard. Since it is expected that all decimal floating-point hardware implementations will conform to the IEEE 754-2008 standard, binding to this standard directly benefits both implementers and programmers.

iTeh STANDARD PREVIEW
(standards.iteh.ai)
ISO/IEC TR 24733:2011
https://standards.iteh.ai/catalog/standards/sist/6b27fd3a-b59c-42da-ab87-10c9fd07faef/iso-iec-tr-24733-2011

© ISO/IEC 2011 – All rights reserved

Information technology — Programming languages, their environments and system software interfaces — Extensions for the programming language C++ to support decimal floating-point arithmetic

1 Scope

This Technical Report specifies an extension to the programming language C++, specified by ISO/IEC 14882:2003. The extension provides support for decimal floating-point arithmetic that is consistent with the specification in IEEE 754-2008. Any conflict between the requirements described here and that specification is unintentional.

The binary floating-point arithmetic specified in IEEE 754-2008 is not considered in this Technical Report.

2 Normative references

iTeh STANDARD PREVIEW

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

(standards.iteh.ai)

ISO/IEC TR 24733:2011
https://standards.iteh.ai/catalog/standards/sist/6b27fd3a-b59c-42da-ab87-10c9fd07faeff/iso-iec-tr-24733-2011

ISO/IEC 14882:2003, *Programming languages — C++*

ISO/IEC TR 19768:2007, *Information technology — Programming languages — Technical Report on C++ Library Extensions*

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*

IEEE 754-2008, *IEEE Standard for Floating-Point Arithmetic*

© ISO/IEC 2011 – All rights reserved

3 Conventions

3.1 General

This Technical Report is non-normative; the extensions that it describes may be considered for inclusion in a future revision of the International Standard for C++, but they are not currently required for conformance to that standard. Furthermore, it is conceivable that a future revision of the International Standard will include facilities that are similar and not identical to the extensions described in this report,

Although this report describes extensions to the *C++ standard library*, vendors may choose to implement these extensions in the C++ language translator itself. This practice is permitted so long as all well-formed programs are accepted by the implementation, and the semantics of those programs which do not have undefined or implementation-defined behavior are as if the extensions had taken the form of a library. [Note: This allows, for instance, an implementation to produce a different result when the extension is implemented in the C++ language translator, for programs that are ill-formed when the extension is implemented as a library.]

The result of deriving a user-defined type from `std::decimal::decimal32`, `std::decimal::decimal64`, or `std::decimal::decimal128` is undefined.

3.2 Relation to C++ Standard Library Introduction

iTeh STANDARD PREVIEW
(standards.iteh.ai)

Unless otherwise specified, the whole of the ISO C++ Standard Library introduction [lib.library] as specified in ISO/IEC 14882 is included into this Technical Report by reference.

ISO/IEC TR 24733:2011
https://standards.iteh.ai/catalog/standards/sist/...
ISO/IEC TR 24733:2011

This Technical Report introduces the following elements to supplement those described in [lib.structure.specifications] as specified in ISO/IEC 14882:

Expansion: the semantics of a macro's expansion text
Library Overload Set: an overload set for a given function name or overloaded operator. Each library overload set specifies the parameter types for the required overloads, and can be implemented as a suitably constrained function template, a set of overloaded non-template functions, or a combination of both.

[*Example:*

Consider a library overload set with the following specification:

```
decimal128 foo(decimal128 d, RHS rhs);
```

> **Library Overload Set:** *RHS* may be any of the decimal floating-point types.
> **Returns:** *d*.

A possible implementation of this library overload set is:

```
decimal128 foo(decimal128 d, decimal128) { return d; }
decimal128 foo(decimal128 d, decimal64)  { return d; }
decimal128 foo(decimal128 d, decimal32)  { return d; }
```


© ISO/IEC 2011 – All rights reserved

Another possible implementation of this library overload set is:

```
template <typename RHS>
decimal128 foo(decimal128 d, RHS) { return d; }
```

In the latter implementation, the template type parameter *RHS* must be constrained to the decimal floating-point types.

--*end example*]

3.3 Relation to "Technical Report on C++ Library Extensions"

Unless otherwise specified, the following sections of ISO/IEC Technical Report 19768: "Technical Report on C++ Library Extensions" are included into this Technical Report by reference:

- General [tr.intro]
- Metaprogramming and type traits [tr.meta]
- Additions to header `<functional>` synopsis [tr.unord.fun.syn]
- Class template `hash` [tr.unord.hash]
- C compatibility [tr.c99]

3.4 Categories of extensions

iTeh STANDARD PREVIEW
(standards.iteh.ai)

This technical report describes 4 categories of library extensions:

- New library components (types and functions) that are declared entirely in new headers, such as the type `decimal::decimal32` in the `<decimal>` header.
- New library components declared as additions to existing standard headers, such as the functions added to the headers `<cmath>` and `<math.h>` in subclause 4.6.
- New library components declared as additions to TR1 headers, such as the template `is_decimal_floating_point` added to the header `<type_traits>` in subclause 4.11.
- Additions to standard library components, such as the specializations of `std::numeric_limits` in subclause 4.3.

ISO/IEC TR 24733:2011
https://standards.iteh.ai/catalog/standards/sist/6b27fd3a-b59c-42da-ab87-10c9fd07fae0/iso-iec-tr-24733-2011

New headers are distinguished from extensions to existing headers by the title of the *synopsis* clause. In the first case the title is of the form "Header `<foo>` synopsis," and the synopsis includes all namespace scope declarations contained in the header. In the second case the title is of the form "Additions to header `<foo>` synopsis" and the synopsis includes only the extensions, *i.e.* those namespace scope declarations that are not present in ISO/IEC 14882 or ISO/IEC 19768.

© ISO/IEC 2011 – All rights reserved

3.5 Namespaces and headers

The extensions described in this technical report are declared within the namespace `decimal`, which is nested inside the namespace `std`.

Unless otherwise specified, references to other entities described in this technical report are assumed to be qualified with `std::decimal::`, references to entities described in the C++ standard library are assumed to be qualified with `std::`, and references to entities described in TR1 are assumed to be qualified with `std::tr1::`.

Even when an extension is specified as additions to standard headers (the second and third categories in section 3.4), vendors should not simply add declarations to standard headers in a way that would be visible to users by default [*Note:* That would fail to be standard conforming, because the new names, even within a namespace, could conflict with user macros. --*end note*] Users should be required to take explicit action to have access to library extensions. It is recommended either that additional declarations in standard headers be protected with a macro that is not defined by default, or else that all extended headers, including both new headers and parallel versions of standard headers with nonstandard declarations, be placed in a separate directory that is not part of the default search path.

iTeh STANDARD PREVIEW
(standards.iteh.ai)
ISO/IEC TR 24733:2011
https://standards.iteh.ai/catalog/standards/sist/6b27fd3a-b59c-42da-ab87-10c9fd07faef/iso-iec-tr-24733-2011

© ISO/IEC 2011 – All rights reserved

4 Decimal floating-point types

4.1 Characteristics of decimal floating-point types

This Technical Report introduces three *decimal floating-point types*, named decimal32, decimal64, and decimal128. The set of values of type decimal32 is a subset of the set of values of type decimal64; the set of values of the type decimal64 is a subset of the set of values of the type decimal128. These types supplement the standard C++ types `float`, `double`, and `long double`, which are collectively described as the *basic floating types*.

The three decimal encoding formats defined in IEEE 754-2008 correspond to the three decimal floating types as follows:

- decimal32 is a *decimal32* number, which is encoded in four consecutive octets (32 bits)
- decimal64 is a *decimal64* number, which is encoded in eight consecutive octets (64 bits)
- decimal128 is a *decimal128* number, which is encoded in 16 consecutive octets (128 bits)

The value of a finite number is given by $(-1)^{sign}$ x coefficient x $10^{exponent}$. Refer to IEEE 754-2008 for details of the format.

iTeh STANDARD PREVIEW

These formats are characterized by the length of the coefficient, and the maximum and minimum exponent. The coefficient is not normalized, so trailing zeros are significant; i.e. 1.0 is equal to but can be distinguished from 1.00. Table 1 shows these characteristics by format.

(standards.iteh.ai)
ISO/IEC TR 24733:2011
https://standards.iteh.ai/catalog/standards/sist/6b27fd3a-b59c-42da-ab87-10c9fd07faef/iso-iec-tr-24733-2011

Table 1 -- Format Characteristics

Format	**decimal32**	**decimal64**	**decimal128**
Coefficient length in digits	7	16	34
Maximum Exponent (E^{max})	97	385	6145
Minimum Exponent (E^{min})	-94	-382	-6142

© ISO/IEC 2011 – All rights reserved

4.2 Decimal Types

4.2.1 Header <decimal> synopsis

```
#include <iosfwd>

namespace std {
namespace decimal {

  // 4.2.2 class decimal32:
  class decimal32;

  // 4.2.3 class decimal64:
  class decimal64;

  // 4.2.4 class decimal128:
  class decimal128;

  // 4.2.5 initialization from coefficient and exponent:
  decimal32  make_decimal32 (long long coeff, int exponent);
  decimal32  make_decimal32 (unsigned long long coeff,
                             int exponent);
  decimal64  make_decimal64 (long long coeff, int exponent);
  decimal64  make_decimal64 (unsigned long long coeff,
                             int exponent);
  decimal128 make_decimal128(long long coeff, int exponent);
  decimal128 make_decimal128(unsigned long long coeff,
                             int exponent);

  // 4.2.6 conversion to generic floating-point type:
  float decimal32_to_float (decimal32 d);
  float decimal64_to_float (decimal64 d);
  float decimal128_to_float(decimal128 d);
  float decimal_to_float(decimal32  d);
  float decimal_to_float(decimal64  d);
  float decimal_to_float(decimal128 d);

  double decimal32_to_double (decimal32 d);
  double decimal64_to_double (decimal64 d);
  double decimal128_to_double(decimal128 d);
  double decimal_to_double(decimal32  d);
  double decimal_to_double(decimal64  d);
  double decimal_to_double(decimal128 d);

  long double decimal32_to_long_double (decimal32 d);
  long double decimal64_to_long_double (decimal64 d);
  long double decimal128_to_long_double(decimal128 d);
  long double decimal_to_long_double(decimal32 d);
  long double decimal_to_long_double(decimal64 d);
  long double decimal_to_long_double(decimal128 d);


  // 4.2.7 unary arithmetic operators:
  decimal32  operator+(decimal32  rhs);
```

iTeh STANDARD PREVIEW
(standards.iteh.ai)
ISO/IEC TR 24733:2011
https://standards.iteh.ai/catalog/standards/sist/6b27fd3a-b59c-42da-ab87-f0c9fd07fdef/iso-iec-tr-24733-2011

© ISO/IEC 2011 – All rights reserved

```
decimal64  operator+(decimal64  rhs);
decimal128 operator+(decimal128 rhs);
decimal32  operator-(decimal32  rhs);
decimal64  operator-(decimal64  rhs);
decimal128 operator-(decimal128 rhs);

// 4.2.8 binary arithmetic operators:
/* see 4.2.8 */ operator+(LHS lhs, decimal32  rhs);
/* see 4.2.8 */ operator+(LHS lhs, decimal64  rhs);
decimal128      operator+(LHS lhs, decimal128 rhs);

decimal32  operator+(decimal32  lhs, RHS rhs);
decimal64  operator+(decimal64  lhs, RHS rhs);
decimal128 operator+(decimal128 lhs, RHS rhs);

/* see 4.2.8 */ operator-(LHS lhs, decimal32  rhs);
/* see 4.2.8 */ operator-(LHS lhs, decimal64  rhs);
decimal128      operator-(LHS lhs, decimal128 rhs);

decimal32  operator-(decimal32  lhs, RHS rhs);
decimal64  operator-(decimal64  lhs, RHS rhs);
decimal128 operator-(decimal128 lhs, RHS rhs);

/* see 4.2.8 */ operator*(LHS lhs, decimal32  rhs);
/* see 4.2.8 */ operator*(LHS lhs, decimal64  rhs);
decimal128      operator*(LHS lhs, decimal128 rhs);

decimal32  operator*(decimal32  lhs, RHS rhs);
decimal64  operator*(decimal64  lhs, RHS rhs);
decimal128 operator*(decimal128 lhs, RHS rhs);

/* see 4.2.8 */ operator/(LHS lhs, decimal32  rhs);
/* see 4.2.8 */ operator/(LHS lhs, decimal64  rhs);
decimal128      operator/(LHS lhs, decimal128 rhs);

decimal32  operator/(decimal32  lhs, RHS rhs);
decimal64  operator/(decimal64  lhs, RHS rhs);
decimal128 operator/(decimal128 lhs, RHS rhs);


// 4.2.9 comparison operators:
bool operator==(LHS lhs, decimal32  rhs);
bool operator==(LHS lhs, decimal64  rhs);
bool operator==(LHS lhs, decimal128 rhs);

bool operator==(decimal32  lhs, RHS rhs);
bool operator==(decimal64  lhs, RHS rhs);
bool operator==(decimal128 lhs, RHS rhs);

bool operator!=(LHS lhs, decimal32  rhs);
bool operator!=(LHS lhs, decimal64  rhs);
bool operator!=(LHS lhs, decimal128 rhs);

bool operator!=(decimal32  lhs, RHS rhs);
```

iTeh STANDARD PREVIEW
(standards.iteh.ai)
ISO/IEC TR 24733:2011
https://standards.iteh.ai/catalog/standards/sist/6b27fd3a-b59c-43da-ab87-10c9fd07faef/iso-iec-tr-24733-2011

© ISO/IEC 2011 – All rights reserved