



**ISO/IEC JTC 1/SC 22/WG 21 N2717
INCITS/PL22.16 08-0227**

Date: 2008-08-18
ISO/IEC Working Draft 29124
ISO/IEC JTC 1/SC 22/WG 21
Secretariat: ANSI

Information Technology —

Programming languages, environments and system software interfaces —

Extensions to the C++ Library to Support Mathematical Special Functions

Warning

This document is not an ISO International Standard. It is distributed for review and comment. It is subject to change without notice and shall not be referred to as an International Standard.

Recipients of this draft are invited to submit, with their comments, notification of any relevant patent rights of which they are aware and to provide supporting documentation.

Document type: International Standard
Document subtype: n/a
Document stage: (2)
Document language: E

Copyright notice

This ISO document is a Draft International Standard and is copyright-protected by ISO. Except as permitted under the applicable laws of the user's country, neither this ISO draft nor any extract from it may be reproduced, stored in a retrieval system or transmitted in any form or by any means, electronic, photocopying, recording or otherwise, without prior written permission being secured.

Requests for permission to reproduce should be addressed to 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 <http://www.iso.org>

Reproduction may be subject to royalty payments or a licensing agreement.

Violators may be prosecuted.

Contents

Contents	iii
List of Tables	v
Foreword	vi
Introduction	vii
1 Scope	1
1.1 Purpose and intent	1
1.2 Relation to C++ Standard Library Introduction	1
1.3 Categories of extensions	1
1.4 Headers	1
1.5 Namespaces	2
2 Normative references	3
3 Terms, definitions, and symbols	5
4 Conformance	7
5 Macro names	9
5.1 Predefined macro name	9
5.2 User-defined macro name	9
6 Mathematical special functions	11
6.1 Additions to header <code><cmath></code>	11
6.1.1 associated Laguerre polynomials	14
6.1.2 associated Legendre polynomials	14
6.1.3 beta function	15
6.1.4 (complete) elliptic integral of the first kind	15
6.1.5 (complete) elliptic integral of the second kind	15
6.1.6 (complete) elliptic integral of the third kind	15
6.1.7 regular modified cylindrical Bessel functions	16
6.1.8 cylindrical Bessel functions (of the first kind)	16

6.1.9	irregular modified cylindrical Bessel functions	16
6.1.10	cylindrical Neumann functions	17
6.1.11	(incomplete) elliptic integral of the first kind	17
6.1.12	(incomplete) elliptic integral of the second kind	18
6.1.13	(incomplete) elliptic integral of the third kind	18
6.1.14	exponential integral	18
6.1.15	Hermite polynomials	18
6.1.16	Laguerre polynomials	19
6.1.17	Legendre polynomials	19
6.1.18	Riemann zeta function	19
6.1.19	spherical Bessel functions (of the first kind)	20
6.1.20	spherical associated Legendre functions	20
6.1.21	spherical Neumann functions	21
6.2	Additions to header <math.h>	21
6.3	The header <ctgmath>	21
6.4	The header <tgmth.h>	21
Bibliography		23
Index		25

List of Tables

1	Summary of affected headers	1
2	Additions to header <code><cmath></code> synopsis	11

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.

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 29124 was prepared by Joint Technical Committee ISO/IEC JTC 1, *Information technology*, subcommittee SC 22, *Programming languages, their environments and systems software interfaces*. WG 21, the ISO Working Group responsible for this Standard, maintains <http://www.open-std.org/jtc1/sc22/wg21>, a site on the World Wide Web containing additional information relevant to this Standard such as background for many of the decisions made during its preparation and a log of any Defect Reports and their responses.

Introduction

This International Standard is divided into three major subdivisions:

- preliminary elements (Clauses 1–3),
- indicating conformance (Clause 4), and
- the library facilities (Clauses 5–6).

Footnotes are provided to emphasize consequences of the rules described in that subclause or elsewhere in this International Standard. References are used to refer to other related documents and subclauses. A bibliography lists documents that are cited in this Standard, or that were referred to during the preparation of this Standard.

1 Scope

[scope]

1.1 Purpose and intent

[scope.purpose]

- 1 This International Standard describes extensions to the *C++ standard library* as described in the International Standard for the C++ programming language [8].
- 2 The provisions of this International Standard are based on 5.2 (“Mathematical special functions”) of *Technical Report 1 on C++ Library Extensions* [9]. That subclause also served as a basis for a similar International Standard [10] by ISO/IEC JTC 1/SC 22/WG 14. Because of their common origin, it is intended that both these International Standards specify substantially identical syntax and semantics to the extent permitted by each Standard’s programming language.

1.2 Relation to C++ Standard Library Introduction

[scope.libintro]

- 1 Unless otherwise specified, the whole of Chapter 17 (“Library Introduction”) of the ISO C++ Standard [8] is included into this International Standard by reference.

1.3 Categories of extensions

[scope.ext]

- 1 This International Standard describes extensions to the C++ standard library to support mathematical special functions.
- 2 These extensions constitute new library components that are declared as additions to an existing standard header, as specified below. For consistency throughout the C++ standard library, adjustments to additional headers are also specified below. For some of these headers, the adjustments are specified as applicable only if the implementation provides the header.

1.4 Headers

[scope.headers]

- 1 The headers affected by this International Standard are summarized in Table 1.

Table 1: Summary of affected headers

Subclause	Header(s)
6.1	<cmath>
6.2	<math.h>
6.3	<ctgmath>
6.4	<tgmath.h>

- 2 Vendors should not simply add declarations to standard headers in a way that would be visible to users by default.¹⁾ Users should be required to take explicit action to have access to library extensions.

¹⁾That would fail to be standard conforming, even within a namespace, because the new names could conflict with user macros.

- 3 The visibility of the library extensions specified in Clause 6 shall be in accordance with the provisions of subclause 5.2.

1.5 Namespaces**[scope.namespaces]**

- 1 Unless otherwise specified, all components described in this International Standard are declared in namespace `std`.
- 2 Unless otherwise specified, all references to components described in the C++ standard library are assumed to be qualified with `std::`.

2 Normative references

[norm]

- 1 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.
- 2 IEC publication 60559:1989 [1].
- 3 ISO publication 31-11:1992 [2].
- 4 ISO/IEC publications
 - 2382-1:1993 [3],
 - 9899:1999 [4],
 - 9899:1999/Cor 1:2001 [5],
 - 9899:1999/Cor 2:2004 [6],
 - 9899:1999/Cor 3:2007 [7], and
 - 14882:2003 [8].

3 Terms, definitions, and symbols

[terms]

- ¹ For the purposes of this document, the terms and definitions given in ISO/IEC 14882:2003 [8], ISO/IEC 9899:1999 [4], and ISO/IEC 2382-1:1993 [3] apply. Other terms are defined where they appear in *italic* type.

4 Conformance

[confor]

- 1 If a "shall" requirement is violated, the behavior is undefined.
- 2 The constraints upon, and latitude of, implementations of the extensions specified in this International Standard are identical to those specified in Chapter 17 ("Library Introduction") of the ISO C++ Standard [8], previously (1.2) included into this International Standard by reference.

5 Macro names

[macro]

5.1 Predefined macro name

[macro.pre]

- 1 The following macro name shall be conditionally defined by the implementation:

`__CPP_MATH_SPEC_FUNCS` The value 200809L, intended to indicate conformance to this International Standard.²⁾

5.2 User-defined macro name

[macro.user]

- 1 The functions declared or defined in Clause 6 are not declared or defined by their respective headers if `__CPP_WANT_MATH_SPEC_FUNCS` is defined as a macro that expands to the integer constant 0 at the point in the source file where the appropriate header is included.
- 2 The functions declared or defined in Clause 6 are declared and defined by their respective headers if `__CPP_WANT_MATH_SPEC_FUNCS` is defined as a macro that expands to the integer constant 1 at the point in the source file where the appropriate header is included.
- 3 It is implementation-defined whether the functions declared or defined in Clause 6 are declared or defined by their respective headers if `__CPP_WANT_MATH_SPEC_FUNCS` is defined as a macro that expands to an integer constant other than 0 or 1 at the point in the source file where the appropriate header is included.³⁾
- 4 It is implementation-defined whether the functions declared or defined in Clause 6 are declared or defined by their respective headers if `__CPP_WANT_MATH_SPEC_FUNCS` is not defined as a macro at the point in the source file where the appropriate header is included.
- 5 Within a preprocessing translation unit, `__CPP_WANT_MATH_SPEC_FUNCS` either shall be nowhere defined as a macro, or else shall be defined identically for all inclusions of any headers from Clause 6. If `__CPP_WANT_MATH_SPEC_FUNCS` is defined differently for any such inclusion, the implementation shall issue a diagnostic as if a preprocessor error directive were used.

²⁾It is intended that future versions of this International Standard will replace the value of this macro with a greater value of type `long int`.

³⁾Future revisions of this International Standard may define meanings for other values of `__CPP_WANT_MATH_SPEC_FUNCS`.

6 Mathematical special functions

[sf]

6.1 Additions to header <cmath>

[sf.cmath]

- 1 Table 2 summarizes the functions that are added to header <cmath>.

Table 2: Additions to header <cmath> synopsis

Functions:		
assoc_laguerre	cyl_bessel_j	hermite
assoc_legendre	cyl_bessel_k	legendre
beta	cyl_neumann	laguerre
comp_ellint_1	ellint_1	riemann_zeta
comp_ellint_2	ellint_2	sph_bessel
comp_ellint_3	ellint_3	sph_legendre
cyl_bessel_i	expint	sph_neumann

- 2 Each of these functions is provided for arguments of types float, double, and long double. The detailed signatures added to header <cmath> are:

// [6.1.1] associated Laguerre polynomials:

```
double      assoc_laguerre(unsigned n, unsigned m, double x);
float       assoc_laguerref(unsigned n, unsigned m, float x);
long double assoc_laguerrel(unsigned n, unsigned m, long double x);
```

// [6.1.2] associated Legendre polynomials:

```
double      assoc_legendre(unsigned l, unsigned m, double x);
float       assoc_legendref(unsigned l, unsigned m, float x);
long double assoc_legendrel(unsigned l, unsigned m, long double x);
```

// [6.1.3] beta function:

```
double      beta(double x, double y);
float       betaf(float x, float y);
long double betal(long double x, long double y);
```

// [6.1.4] (complete) elliptic integral of the first kind:

```
double      comp_ellint_1(double k);
float       comp_ellint_1f(float k);
long double comp_ellint_1l(long double k);
```

```
// [6.1.5] (complete) elliptic integral of the second kind:
double      comp_ellint_2(double k);
float       comp_ellint_2f(float k);
long double comp_ellint_2l(long double k);

// [6.1.6] (complete) elliptic integral of the third kind:
double      comp_ellint_3(double k, double nu);
float       comp_ellint_3f(float k, float nu);
long double comp_ellint_3l(long double k, long double nu);

// [6.1.7] regular modified cylindrical Bessel functions:
double      cyl_bessel_i(double nu, double x);
float       cyl_bessel_if(float nu, float x);
long double cyl_bessel_il(long double nu, long double x);

// [6.1.8] cylindrical Bessel functions (of the first kind):
double      cyl_bessel_j(double nu, double x);
float       cyl_bessel_jf(float nu, float x);
long double cyl_bessel_jl(long double nu, long double x);

// [6.1.9] irregular modified cylindrical Bessel functions:
double      cyl_bessel_k(double nu, double x);
float       cyl_bessel_kf(float nu, float x);
long double cyl_bessel_kl(long double nu, long double x);

// [6.1.10] cylindrical Neumann functions;
// cylindrical Bessel functions (of the second kind):
double      cyl_neumann(double nu, double x);
float       cyl_neumannf(float nu, float x);
long double cyl_neumannl(long double nu, long double x);

// [6.1.11] (incomplete) elliptic integral of the first kind:
double      ellint_1(double k, double phi);
float       ellint_1f(float k, float phi);
long double ellint_1l(long double k, long double phi);

// [6.1.12] (incomplete) elliptic integral of the second kind:
double      ellint_2(double k, double phi);
float       ellint_2f(float k, float phi);
long double ellint_2l(long double k, long double phi);

// [6.1.13] (incomplete) elliptic integral of the third kind:
double      ellint_3(double k, double nu, double phi);
float       ellint_3f(float k, float nu, float phi);
long double ellint_3l(long double k, long double nu, long double phi);

// [6.1.14] exponential integral:
double      expint(double x);
float       expintf(float x);
long double expintl(long double x);
```

```

// [6.1.15] Hermite polynomials:
double      hermite(unsigned n, double x);
float       hermitef(unsigned n, float x);
long double hermitel(unsigned n, long double x);

// [6.1.16] Laguerre polynomials:
double      laguerre(unsigned n, double x);
float       laguerref(unsigned n, float x);
long double laguerrel(unsigned n, long double x);

// [6.1.17] Legendre polynomials:
double      legendre(unsigned l, double x);
float       legendref(unsigned l, float x);
long double legendrel(unsigned l, long double x);

// [6.1.18] Riemann zeta function:
double      riemann_zeta(double x);
float       riemann_zetaf(float x);
long double riemann_zetal(long double x);

// [6.1.19] spherical Bessel functions (of the first kind):
double      sph_bessel(unsigned n, double x);
float       sph_besself(unsigned n, float x);
long double sph_bessell(unsigned n, long double x);

// [6.1.20] spherical associated Legendre functions:
double      sph_legendre(unsigned l, unsigned m, double theta);
float       sph_legendref(unsigned l, unsigned m, float theta);
long double sph_legendrel(unsigned l, unsigned m, long double theta);

// [6.1.21] spherical Neumann functions;
// spherical Bessel functions (of the second kind):
double      sph_neumann(unsigned n, double x);
float       sph_neumannf(unsigned n, float x);
long double sph_neumannl(unsigned n, long double x);

```

- 3 Each of the functions specified above that has one or more double parameters (the double version) shall have two additional overloads:

1. a version with each double parameter replaced with a float parameter (the float version), and
2. a version with each double parameter replaced with a long double parameter (the long double version).

The return type of each such float version shall be `float`, and the return type of each such long double version shall be `long double`.

- 4 Moreover, each double version shall have sufficient additional overloads to determine which of the above three versions to actually call, by the following ordered set of rules:
1. First, if any argument corresponding to a double parameter in the double version has type `long double`, the `long double` version is called.

2. Otherwise, if any argument corresponding to a `double` parameter in the `double` version has type `double` or has an integer type, the `double` version is called.
 3. Otherwise, the `float` version is called.
- 5 If any argument value to any of the functions specified above is a NaN (Not a Number), the function shall return a NaN but it shall not report a domain error. Otherwise, the function shall report a domain error for just those argument values for which:
- a) the function description's *Returns* clause explicitly specifies a domain and those argument values fall outside the specified domain, or
 - b) the corresponding mathematical function value has a non-zero imaginary component, or
 - c) the corresponding mathematical function is not mathematically defined.⁴⁾
- 6 Unless otherwise specified, each function is defined for all finite values, for negative infinity, and for positive infinity.

6.1.1 associated Laguerre polynomials

[sf.cmath.Lnm]

```
double    assoc_laguerre(unsigned n, unsigned m, double x);
float     assoc_laguerref(unsigned n, unsigned m, float x);
long double assoc_laguerrel(unsigned n, unsigned m, long double x);
```

- 1 *Effects:* These functions compute the associated Laguerre polynomials of their respective arguments `n`, `m`, and `x`.
- 2 *Returns:* The `assoc_laguerre` functions return

$$L_n^m(x) = (-1)^m \frac{d^m}{dx^m} L_{n+m}(x), \quad \text{for } x \geq 0.$$

- 3 *Remark:* The effect of calling each of these functions is implementation-defined if `n` \geq 128 or if `m` \geq 128.

6.1.2 associated Legendre polynomials

[sf.cmath.Plm]

```
double    assoc_legendre(unsigned l, unsigned m, double x);
float     assoc_legendref(unsigned l, unsigned m, float x);
long double assoc_legendrel(unsigned l, unsigned m, long double x);
```

- 1 *Effects:* These functions compute the associated Legendre functions of their respective arguments `l`, `m`, and `x`.
- 2 *Returns:* The `assoc_legendre` functions return

$$P_\ell^m(x) = (1-x^2)^{m/2} \frac{d^m}{dx^m} P_\ell(x), \quad \text{for } |x| \leq 1.$$

- 3 *Remark:* The effect of calling each of these functions is implementation-defined if `l` \geq 128.

⁴⁾A mathematical function is mathematically defined for a given set of argument values (a) if it is explicitly defined for that set of argument values, or (b) if its limiting value exists and does not depend on the direction of approach.

6.1.3 beta function**[sf.cmath.beta]**

```
double    beta(double x, double y);
float     betaf(float x, float y);
long double betal(long double x, long double y);
```

1 *Effects:* These functions compute the beta function of their respective arguments x and y .

2 *Returns:* The beta functions return

$$B(x,y) = \frac{\Gamma(x)\Gamma(y)}{\Gamma(x+y)}, \quad \text{for } x > 0, y > 0.$$

6.1.4 (complete) elliptic integral of the first kind**[sf.cmath.elK]**

```
double    comp_ellint_1(double k);
float     comp_ellint_1f(float k);
long double comp_ellint_1l(long double k);
```

1 *Effects:* These functions compute the complete elliptic integral of the first kind of their respective arguments k .

2 *Returns:* The `comp_ellint_1` functions return

$$K(k) = F(k, \pi/2), \quad \text{for } |k| \leq 1.$$

3 See also [6.1.11](#).

6.1.5 (complete) elliptic integral of the second kind**[sf.cmath.elEX]**

```
double    comp_ellint_2(double k);
float     comp_ellint_2f(float k);
long double comp_ellint_2l(long double k);
```

1 *Effects:* These functions compute the complete elliptic integral of the second kind of their respective arguments k .

2 *Returns:* The `comp_ellint_2` functions return

$$E(k) = E(k, \pi/2), \quad \text{for } |k| \leq 1.$$

3 See also [6.1.12](#).

6.1.6 (complete) elliptic integral of the third kind**[sf.cmath.elPx]**

```
double    comp_ellint_3(double k, double nu);
float     comp_ellint_3f(float k, float nu);
long double comp_ellint_3l(long double k, long double nu);
```

1 *Effects:* These functions compute the complete elliptic integral of the third kind of their respective arguments k and ν .

2 *Returns:* The `comp_ellint_3` functions return

$$\Pi(\nu, k) = \Pi(\nu, k, \pi/2), \quad \text{for } |k| \leq 1.$$

3 See also [6.1.13](#).

6.1.7 regular modified cylindrical Bessel functions

[sf.cmath.I]

```
double    cyl_bessel_i(double nu, double x);
float     cyl_bessel_if(float nu, float x);
long double cyl_bessel_il(long double nu, long double x);
```

1 *Effects:* These functions compute the regular modified cylindrical Bessel functions of their respective arguments `nu` and `x`.

2 *Returns:* The `cyl_bessel_i` functions return

$$I_\nu(x) = i^{-\nu} J_\nu(ix) = \sum_{k=0}^{\infty} \frac{(x/2)^{\nu+2k}}{k! \Gamma(\nu+k+1)}, \quad \text{for } x \geq 0.$$

3 *Remark:* The effect of calling each of these functions is implementation-defined if `nu >= 128`.

4 See also [6.1.8](#).

6.1.8 cylindrical Bessel functions (of the first kind)

[sf.cmath.J]

```
double    cyl_bessel_j(double nu, double x);
float     cyl_bessel_jf(float nu, float x);
long double cyl_bessel_jl(long double nu, long double x);
```

1 *Effects:* These functions compute the cylindrical Bessel functions of the first kind of their respective arguments `nu` and `x`.

2 *Returns:* The `cyl_bessel_j` functions return

$$J_\nu(x) = \sum_{k=0}^{\infty} \frac{(-1)^k (x/2)^{\nu+2k}}{k! \Gamma(\nu+k+1)}, \quad \text{for } x \geq 0.$$

3 *Remark:* The effect of calling each of these functions is implementation-defined if `nu >= 128`.

6.1.9 irregular modified cylindrical Bessel functions

[sf.cmath.K]

```
double    cyl_bessel_k(double nu, double x);
float     cyl_bessel_kf(float nu, float x);
long double cyl_bessel_kl(long double nu, long double x);
```


1 *Effects:* These functions compute the irregular modified cylindrical Bessel functions of their respective arguments `nu` and `x`.

2 *Returns:* The `cyl_bessel_k` functions return

$$K_\nu(x) = (\pi/2)i^{\nu+1}(J_\nu(ix) + iN_\nu(ix)) = \begin{cases} \frac{\pi}{2} \frac{I_{-\nu}(x) - I_\nu(x)}{\sin \nu\pi}, & \text{for } x \geq 0 \text{ and non-integral } \nu \\ \frac{\pi}{2} \lim_{\mu \rightarrow \nu} \frac{I_{-\mu}(x) - I_\mu(x)}{\sin \mu\pi}, & \text{for } x \geq 0 \text{ and integral } \nu \end{cases}$$

3 *Remark:* The effect of calling each of these functions is implementation-defined if `nu` \geq 128.

4 See also [6.1.7](#).

6.1.10 cylindrical Neumann functions

[sf.cmath.N]

```
double    cyl_neumann(double nu, double x);
float     cyl_neumannf(float nu, float x);
long double cyl_neumannl(long double nu, long double x);
```

1 *Effects:* These functions compute the cylindrical Neumann functions, also known as the cylindrical Bessel functions of the second kind, of their respective arguments `nu` and `x`.

2 *Returns:* The `cyl_neumann` functions return

$$N_\nu(x) = \begin{cases} \frac{J_\nu(x) \cos \nu\pi - J_{-\nu}(x)}{\sin \nu\pi}, & \text{for } x \geq 0 \text{ and non-integral } \nu \\ \lim_{\mu \rightarrow \nu} \frac{J_\mu(x) \cos \mu\pi - J_{-\mu}(x)}{\sin \mu\pi}, & \text{for } x \geq 0 \text{ and integral } \nu \end{cases}$$

3 *Remark:* The effect of calling each of these functions is implementation-defined if `nu` \geq 128.

4 See also [6.1.8](#).

6.1.11 (incomplete) elliptic integral of the first kind

[sf.cmath.ellF]

```
double    ellint_1(double k, double phi);
float     ellint_1f(float k, float phi);
long double ellint_1l(long double k, long double phi);
```

1 *Effects:* These functions compute the incomplete elliptic integral of the first kind of their respective arguments `k` and `phi` (`phi` measured in radians).

2 *Returns:* The `ellint_1` functions return

$$F(k, \phi) = \int_0^\phi \frac{d\theta}{\sqrt{1 - k^2 \sin^2 \theta}}, \quad \text{for } |k| \leq 1.$$

6.1.12 (incomplete) elliptic integral of the second kind**[sf.cmath.ellE]**

```
double    ellint_2(double k, double phi);
float     ellint_2f(float k, float phi);
long double ellint_2l(long double k, long double phi);
```

1 *Effects:* These functions compute the incomplete elliptic integral of the second kind of their respective arguments k and ϕ (ϕ measured in radians).

2 *Returns:* The `ellint_2` functions return

$$E(k, \phi) = \int_0^\phi \sqrt{1 - k^2 \sin^2 \theta} \, d\theta, \quad \text{for } |k| \leq 1.$$

6.1.13 (incomplete) elliptic integral of the third kind**[sf.cmath.ellP]**

```
double    ellint_3(double k, double nu, double phi);
float     ellint_3f(float k, float nu, float phi);
long double ellint_3l(long double k, long double nu, long double phi);
```

1 *Effects:* These functions compute the incomplete elliptic integral of the third kind of their respective arguments k , ν , and ϕ (ϕ measured in radians).

2 *Returns:* The `ellint_3` functions return

$$\Pi(\nu, k, \phi) = \int_0^\phi \frac{d\theta}{(1 - \nu \sin^2 \theta) \sqrt{1 - k^2 \sin^2 \theta}}, \quad \text{for } |k| \leq 1.$$

6.1.14 exponential integral**[sf.cmath.ei]**

```
double    expint(double x);
float     expintf(float x);
long double expintl(long double x);
```

1 *Effects:* These functions compute the exponential integral of their respective arguments x .

2 *Returns:* The `expint` functions return

$$\text{Ei}(x) = - \int_{-x}^{\infty} \frac{e^{-t}}{t} \, dt.$$

6.1.15 Hermite polynomials**[sf.cmath.Hn]**

```
double    hermite(unsigned n, double x);
float     hermitef(unsigned n, float x);
long double hermitel(unsigned n, long double x);
```

1 *Effects:* These functions compute the Hermite polynomials of their respective arguments n and x .

2 *Returns:* The hermite functions return

$$H_n(x) = (-1)^n e^{x^2} \frac{d^n}{dx^n} e^{-x^2}.$$

3 *Remark:* The effect of calling each of these functions is implementation-defined if $n \geq 128$.

6.1.16 Laguerre polynomials

[sf.cmath.Ln]

```
double    laguerre(unsigned n, double x);
float     laguerref(unsigned n, float x);
long double laguerrel(unsigned n, long double x);
```

1 *Effects:* These functions compute the Laguerre polynomials of their respective arguments n and x .

2 *Returns:* The laguerre functions return

$$L_n(x) = \frac{e^x}{n!} \frac{d^n}{dx^n} (x^n e^{-x}), \quad \text{for } x \geq 0.$$

3 *Remark:* The effect of calling each of these functions is implementation-defined if $n \geq 128$.

6.1.17 Legendre polynomials

[sf.cmath.Pl]

```
double    legendre(unsigned l, double x);
float     legendref(unsigned l, float x);
long double legendrel(unsigned l, long double x);
```

1 *Effects:* These functions compute the Legendre polynomials of their respective arguments l and x .

2 *Returns:* The legendre functions return

$$P_\ell(x) = \frac{1}{2^\ell \ell!} \frac{d^\ell}{dx^\ell} (x^2 - 1)^\ell, \quad \text{for } |x| \leq 1.$$

3 *Remark:* The effect of calling each of these functions is implementation-defined if $l \geq 128$.

6.1.18 Riemann zeta function

[sf.cmath.riemannzeta]

```
double    riemann_zeta(double x);
float     riemann_zetaf(float x);
long double riemann_zetal(long double x);
```

1 *Effects:* These functions compute the Riemann zeta function of their respective arguments x .

2 *Returns:* The `riemann_zeta` functions return

$$\zeta(x) = \begin{cases} \sum_{k=1}^{\infty} k^{-x}, & \text{for } x > 1 \\ \frac{1}{1-2^{1-x}} \sum_{k=1}^{\infty} (-1)^{k-1} k^{-x}, & \text{for } 0 \leq x \leq 1 \\ 2^x \pi^{x-1} \sin\left(\frac{\pi x}{2}\right) \Gamma(1-x) \zeta(1-x), & \text{for } x < 0 \end{cases} .$$

6.1.19 spherical Bessel functions (of the first kind)

[sf.cmath.j]

```
double    sph_bessel(unsigned n, double x);
float     sph_besself(unsigned n, float x);
long double sph_bessell(unsigned n, long double x);
```

1 *Effects:* These functions compute the spherical Bessel functions of the first kind of their respective arguments `n` and `x`.

2 *Returns:* The `sph_bessel` functions return

$$j_n(x) = (\pi/2x)^{1/2} J_{n+1/2}(x), \quad \text{for } x \geq 0.$$

3 *Remark:* The effect of calling each of these functions is implementation-defined if `n` \geq 128.

4 See also 6.1.8.

6.1.20 spherical associated Legendre functions

[sf.cmath.Ylm]

```
double    sph_legendre(unsigned l, unsigned m, double theta);
float     sph_legendref(unsigned l, unsigned m, float theta);
long double sph_legendrel(unsigned l, unsigned m, long double theta);
```

1 *Effects:* These functions compute the spherical associated Legendre functions of their respective arguments `l`, `m`, and `theta` (`theta` measured in radians).

2 *Returns:* The `sph_legendre` functions return

$$Y_\ell^m(\theta, 0)$$

where

$$Y_\ell^m(\theta, \phi) = (-1)^m \left[\frac{(2\ell+1)}{4\pi} \frac{(\ell-m)!}{(\ell+m)!} \right]^{1/2} P_\ell^m(\cos \theta) e^{im\phi}, \quad \text{for } |m| \leq \ell.$$

3 *Remark:* The effect of calling each of these functions is implementation-defined if `l` \geq 128.

4 See also 6.1.2.

6.1.21 spherical Neumann functions**[sf.cmath.n]**

```
double    sph_neumann(unsigned n, double x);
float     sph_neumannf(unsigned n, float x);
long double sph_neumannl(unsigned n, long double x);
```

1 *Effects:* These functions compute the spherical Neumann functions, also known as the spherical Bessel functions of the second kind, of their respective arguments *n* and *x*.

2 *Returns:* The sph_neumann functions return

$$n_n(x) = (\pi/2x)^{1/2} N_{n+1/2}(x), \quad \text{for } x \geq 0.$$

3 *Remark:* The effect of calling each of these functions is implementation-defined if *n* >= 128.

4 See also [6.1.10](#).

6.2 Additions to header <math.h>**[sf.mathh]**

1 The header <math.h> shall have sufficient additional using declarations to import into the global name space all of the function names specified in the previous section.

6.3 The header <ctgmath>**[sf.ctgmath]**

1 The header <ctgmath>, if provided by the implementation, effectively includes the header <cmath>.

6.4 The header <tgmath.h>**[sf.tgmathh]**

1 The header <tgmath.h>, if provided by the implementation, effectively includes the header <math.h>.

Bibliography

- [1] IEC 60559:1989, *Binary floating-point arithmetic for microprocessor systems*.
- [2] ISO 31-11:1992, *Quantities and units — Part 11: Mathematical signs and symbols for use in the physical sciences and technology*.
- [3] ISO/IEC 2382-1:1993, *Information Technology — Vocabulary — Part 1: Fundamental terms*.
- [4] ISO/IEC 9899:1999, *Programming Languages — C*.
- [5] ISO/IEC 9899:1999/Cor 1:2001, *Programming Languages — C — Technical Corrigendum 1*.
- [6] ISO/IEC 9899:1999/Cor 2:2004, *Programming Languages — C — Technical Corrigendum 2*.
- [7] ISO/IEC 9899:1999/Cor 3:2007, *Programming Languages — C — Technical Corrigendum 3*.
- [8] ISO/IEC 14882:2003, *Programming Languages — C++*.
- [9] ISO/IEC TR 19768:2006, *Technical Report 1 on C++ Library Extensions*.
- [10] ISO/IEC 24747:2008, *Information Technology — Programming languages, environments and system software interfaces — Extensions to the C Library, to Support Mathematical Special Functions*.
- [11] Milton Abramowitz and Irene A. Stegun (eds.), *Handbook of Mathematical Functions with Formulas, Graphs and Mathematical Tables*. National Bureau of Standards Applied Mathematics Series, volume 55. U. S. Government Printing Office, Washington, DC: 1964. Reprinted with corrections, Dover Publications: 1972.

Index

`__CPP_MATH_SPEC_FUNCS`, 9
`__CPP_WANT_MATH_SPEC_FUNCS`, 9

`assoc_laguerre`, 14
`assoc_legendre`, 14

Bessel functions

I_ν , 16
 J_ν , 16
 K_ν , 16
 N_ν , 17
 j_n , 20
 n_n , 21

`beta`, 15
beta functions B , 15

`<cmath>`, 1, 11
`comp_ellint_1`, 15
`comp_ellint_2`, 15
`comp_ellint_3`, 15
`<ctgmath>`, 1, 21
`cyl_bessel_i`, 16
`cyl_bessel_j`, 16
`cyl_bessel_k`, 16
`cyl_neumann`, 17

domain error, 14

E (complete elliptic integrals), 15
 E (incomplete elliptic integrals), 18
 Ei (exponential integrals), 18
`ellint_1`, 17
`ellint_2`, 18
`ellint_3`, 18
elliptic integrals
 complete Π , 15

complete E , 15
complete K , 15
incomplete Π , 18
incomplete E , 18
incomplete F , 17

Eulerian integral of the first kind, *see* beta
`expint`, 18
exponential integrals Ei , 18

F (incomplete elliptic integrals), 17

H_n (Hermite polynomials), 18
`hermite`, 18
Hermite polynomials H_n , 18

I_ν (Bessel functions), 16

j_n (spherical Bessel functions), 20
 J_ν (Bessel functions), 16

K (complete elliptic integrals), 15
 K_ν (Bessel functions), 16

L_n (Laguerre polynomials), 19
 L_n^m (associated Laguerre polynomials), 14
`laguerre`, 19

Laguerre polynomials L_n , 19
Laguerre polynomials L_n^m , 14
`legendre`, 19

Legendre functions Y_ℓ^m , 20
Legendre polynomials P_ℓ , 19
Legendre polynomials P_ℓ^m , 14

macro names
 predefined, 9
 user-defined, 9

`<math.h>`, 1, 21

n_n (spherical Neumann functions), 21

N_ν (Neumann functions), 17

namespaces, 2

NaN, 14

Neumann functions

N_ν , 17

n_n , 21

overloads

 floating point, 13

P_ℓ (Legendre polynomials), 19

P_ℓ^m (associated Legendre polynomials), 14

Π (complete elliptic integrals), 15

Π (incomplete elliptic integrals), 18

`riemann_zeta`, 19

special functions, 11–21

`sph_bessel`, 20

`sph_legendre`, 20

`sph_neumann`, 21

 spherical harmonics Y_ℓ^m , 20

 std namespace, 2

`<tgmath.h>`, 1, 21

Y_ℓ^m (spherical associated Legendre functions), 20

zeta functions ζ , 19