INTERNATIONAL STANDARD

ISO/IEC 29124

First edition
2010-09-15

Information technology — Programming languages, their environments and system software interfaces — Extensions to the C++ Library to support mathematical special functions

Technologies de l'information — Langages de programmation, leur environnement et interfaces des logiciels de systèmes — Extensions à la bibliothèque C++ pour supporter les fonctions mathématiques spéciales

Reference number
ISO/IEC 29124:2010(E)

Contents

List of Tables

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 system software interfaces*.

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 non-normatively cited in this International Standard, or that were referred to during its preparation.

The provisions of this International Standard are based on 5.2 (“Mathematical special functions”) of ISO/IEC TR 19768:2007 [2]. That subclause also served as a basis for a similar International Standard [3]. 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.

Information technology — Programming languages, their environments and system software interfaces — Extensions to the C++ Library to support mathematical special functions

1 Scope [scope]

1.1 Purpose and intent [scope.purpose]

1 This International Standard specifies extensions to the *C++ standard library* as defined in the International Standard for the C++ programming language [ISO/IEC 14882:2003].

1.2 Relation to C++ Standard Library Introduction [scope.libintro]

1 Unless otherwise specified, the whole of Clause 17 (“Library Introduction”) of the ISO C++ Standard [ISO/IEC 14882:2003] is included into this International Standard by reference.

1.3 Categories of extensions [scope.ext]

1 This International Standard specifies extensions to the C++ standard library to support mathematical special functions.

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.

- ISO/IEC 2382-1:1993, *Information technology — Vocabulary — Part 1: Fundamental terms*
- ISO/IEC 9899:1999, *Programming languages — C*
- ISO/IEC 9899:1999/Cor 1:2001, *Programming languages — C — Technical Corrigendum 1*
- ISO/IEC 9899:1999/Cor 2:2004, *Programming languages — C — Technical Corrigendum 2*
- ISO/IEC 9899:1999/Cor 3:2007, *Programming languages — C — Technical Corrigendum 3*
- ISO/IEC 14882:2003, *Programming languages — C++*
- ISO 80000-2:2009, *Quantities and units — Part 2: Mathematical signs and symbols to be used in the natural sciences and technology*

3 Terms, definitions, and symbols [terms]

1 For the purposes of this document, the terms and definitions given in ISO/IEC 14882:2003, ISO/IEC 9899:1999 and its Technical Corrigenda, and ISO/IEC 2382-1:1993 apply, as do the terms, definitions, and symbols given in ISO 80000-2:2009. 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 Clause 17 ("Library Introduction") of the ISO C++ Standard [ISO/IEC 14882:2003], previously (1.2) included into this International Standard by reference.

5 Headers [head]

1 The extensions specified by this International Standard 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.

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

Table 1: Summary of affected headers

Subclause	Header(s)
8.1	`<cmath>`
8.2	`<math.h>`
8.3	`<ctgmath>`
8.4	`<tgmath.h>`

3 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.

4 The visibility of the library extensions specified in Clause 8 shall be in accordance with the provisions of subclause 7.2.

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

6 Namespaces [ns]

1 Unless otherwise specified, all components specified in this International Standard are declared in namespace `std`.

2 Unless otherwise specified, all references to components specified in the C++ standard library are assumed to be qualified with `std::`.

7 Macro names [macro]

7.1 Predefined macro name [macro.pre]

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

`__STDCPP_MATH_SPEC_FUNCS__` The value `201003L`, intended to indicate conformance to this International Standard.[2)]

7.2 User-defined macro name [macro.user]

1 The functions declared or defined in Clause 8 are not declared or defined by their respective headers if `__STDCPP_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 8 are declared and defined by their respective headers if `__STDCPP_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 8 are declared or defined by their respective headers if `__STDCPP_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 8 are declared or defined by their respective headers if `__STDCPP_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, `__STDCPP_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 8. If `__STDCPP_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 specify meanings for other values of `__STDCPP_WANT_MATH_SPEC_FUNCS__`.

8 Mathematical special functions [sf]

8.1 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:

```
// [8.1.1] 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);

// [8.1.2] 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);

// [8.1.3] beta function:
double      beta(double x, double y);
float       betaf(float x, float y);
long double betal(long double x, long double y);

// [8.1.4] (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);
```

```
// [8.1.5] (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);

// [8.1.6] (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);

// [8.1.7] 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);

// [8.1.8] 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);

// [8.1.9] 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);

// [8.1.10] 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);

// [8.1.11] (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);

// [8.1.12] (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);

// [8.1.13] (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);

// [8.1.14] exponential integral:
double       expint(double x);
float        expintf(float x);
long double  expintl(long double x);
```

```
// [8.1.15] Hermite polynomials:
double       hermite(unsigned n, double x);
float        hermitef(unsigned n, float x);
long double  hermitel(unsigned n, long double x);

// [8.1.16] Laguerre polynomials:
double       laguerre(unsigned n, double x);
float        laguerref(unsigned n, float x);
long double  laguerrel(unsigned n, long double x);

// [8.1.17] Legendre polynomials:
double       legendre(unsigned l, double x);
float        legendref(unsigned l, float x);
long double  legendrel(unsigned l, long double x);

// [8.1.18] Riemann zeta function:
double       riemann_zeta(double x);
float        riemann_zetaf(float x);
long double  riemann_zetal(long double x);

// [8.1.19] 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);

// [8.1.20] 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);

// [8.1.21] 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.

8.1.1 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

$$\mathsf{L}_n^m(x) = (-1)^m \frac{\mathsf{d}^m}{\mathsf{d}x^m} \mathsf{L}_{n+m}(x), \quad \text{for } x \geq 0.$$

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

8.1.2 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

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

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

8.1.3 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

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

8.1.4 (complete) elliptic integral of the first kind [sf.cmath.ellK]

```
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

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

3 See also 8.1.11.

8.1.5 (complete) elliptic integral of the second kind [sf.cmath.ellEx]

```
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

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

3 See also 8.1.12.

8.1.6 (complete) elliptic integral of the third kind [sf.cmath.ellPx]

```
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 `nu`.

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 8.1.13.

8.1.7 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

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

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

4 See also 8.1.8.

8.1.8 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

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

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

8.1.9 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

$$\mathsf{K}_\nu(x) = (\pi/2)\mathrm{i}^{\nu+1}(\mathsf{J}_\nu(\mathrm{i}x) + \mathrm{i}\mathsf{N}_\nu(\mathrm{i}x)) = \begin{cases} \dfrac{\pi}{2}\dfrac{\mathsf{I}_{-\nu}(x) - \mathsf{I}_\nu(x)}{\sin\nu\pi}, & \text{for } x \geq 0 \text{ and non-integral } \nu \\ \dfrac{\pi}{2}\lim\limits_{\mu\to\nu}\dfrac{\mathsf{I}_{-\mu}(x) - \mathsf{I}_\mu(x)}{\sin\mu\pi}, & \text{for } x \geq 0 \text{ and integral } \nu \end{cases}.$$

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

4 See also 8.1.7.

8.1.10 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

$$\mathsf{N}_\nu(x) = \begin{cases} \dfrac{\mathsf{J}_\nu(x)\cos\nu\pi - \mathsf{J}_{-\nu}(x)}{\sin\nu\pi}, & \text{for } x \geq 0 \text{ and non-integral } \nu \\ \lim\limits_{\mu\to\nu}\dfrac{\mathsf{J}_\mu(x)\cos\mu\pi - \mathsf{J}_{-\mu}(x)}{\sin\mu\pi}, & \text{for } x \geq 0 \text{ and integral } \nu \end{cases}.$$

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

4 See also 8.1.8.

8.1.11 (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

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

8.1.12 (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 `phi` (`phi` measured in radians).

2 *Returns:* The `ellint_2` functions return

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

8.1.13 (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`, `nu`, and `phi` (`phi` measured in radians).

2 *Returns:* The `ellint_3` functions return

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

8.1.14 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

$$\mathsf{Ei}(x) = -\int_{-x}^{\infty} \frac{e^{-t}}{t}\,\mathrm{d}t\ .$$

8.1.15 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

$$\mathsf{H}_n(x) = (-1)^n e^{x^2} \frac{\mathrm{d}^n}{\mathrm{d}x^n} e^{-x^2} .$$

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

8.1.16 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

$$\mathsf{L}_n(x) = \frac{e^x}{n!} \frac{\mathrm{d}^n}{\mathrm{d}x^n} (x^n e^{-x}), \quad \text{for } x \ge 0.$$

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

8.1.17 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

$$\mathsf{P}_\ell(x) = \frac{1}{2^\ell \, \ell!} \frac{\mathrm{d}^\ell}{\mathrm{d}x^\ell} (x^2 - 1)^\ell, \quad \text{for } |x| \le 1.$$

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

8.1.18 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

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

8.1.19 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

$$\mathrm{j}_n(x) = (\pi/2x)^{1/2} \mathrm{J}_{n+1/2}(x), \quad \text{for } x \ge 0.$$

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

4 See also 8.1.8.

8.1.20 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

$$\mathrm{Y}_\ell^m(\theta, 0)$$

where

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

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

4 See also 8.1.2.

8.1.21 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

$$\mathsf{n}_n(x) = (\pi/2x)^{1/2}\mathsf{N}_{n+1/2}(x), \quad \text{for } x \geq 0.$$

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

4 See also 8.1.10.

8.2 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.

8.3 The header <ctgmath> [sf.ctgmath]

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

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

[3] ISO/IEC 24747:2009, *Information technology — Programming languages, their environments and system software interfaces — Extensions to the C Library to support mathematical special functions*

[4] 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

ICS 35.060
Price based on 21 pages