ISO/IEC JTC 1/SC 22/WG14 N1292

Date: 2008-03-14
ISO/IEC FCD 24747
ISO/IEC JTC 1/SC 22/WG 14
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: (3) Committee Draft International Standard
Document language: E

Copyright notice

Front matter

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 24747 was prepared by Joint Technical Committee ISO/IEC JTC 1, *Information technology*, subcommittee SC22, *Programming languages, their environments and systems software interfaces*.

Contents

List of Tables

1 Scope [intro]

1 This International Standard describes extensions to the *C standard library* that is described in the International Standard for the C programming language [3].

1.1 Relation to C Standard Library Introduction [description]

1 Unless otherwise specified, the whole of the ISO C Standard Library introduction [lib.library] is included into this International Standard by reference.

1.2 Categories of extensions [intro.ext]

1 This International Standard describes library extensions to the C Standard Library to support Mathematical Special functions to be added to <math.h> and <tgmath.h>.

Table 1: Numerical library summary

Subclause	Header(s)
6.2 Additions to	<math.h>
6.3 Additions to	<tgmath.h>

2 Normative references [nor.ref]

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 ISO/IEC 9899:1999, *Programming Languages — C.*

3 ISO/IEC 9899:1999/Cor 1:2001, *Programming Languages —C —Technical Corrigendum 1.*

4 ISO/IEC 9899:1999/Cor 2:2004, *Programming Languages —C —Technical Corrigendum 2.*

5 ISO/IEC 9899:1999/Cor 3:2007, *Programming Languages —C —Technical Corrigendum 3.*

6 ISO 31-11:1992, *Quantities and units —Part 11: Mathematical signs and symbols for use in the physical sciences and technology.*

7 ISO/IEC 2382-1:1993, *Information technology — Vocabulary —- Part 1: Fundamental terms.*

8 IEC 60559:1989, *Binary floating-point arithmetic for microprocessor systems* (previously designated IEC 559:1989).

3 Terms, definitions, and symbols [terms]

1 For the purposes of this document, the terms and definitions given in ISO/IEC 9899:1999 and ISO/IEC 2382-1. Other terms are defined where they appear in *italic* type.

4 Conformance [confor]

1 If a "shall" requirement is violated, the behavior is undefined.

5 Predefined macro names [pre.macro]

1 The following macro name is conditionally defined by the implementation:
`__STDC_MATH_SPEC_FUNCS__` The integer constant `200802`, intended to indicate conformance to this International Standard. [1)]

[1)] The intention is that this will remain an integer constant of type `long int` that is increased with each revision of this International Standard.

6 Mathematical special functions [num.sf]

6.1 Standard headers [num.sf.header]

1 The functions declared in Clause 6 and its subclauses are not declared by their respetive header if `__STDC_WANT_MATH_SPEC_FUNCS__` is defined as a macro which expands to the interger constant `0` at the point in the source file where the appropriate header is included.

2 The functions declared in Clause 6 and its subclauses are declared by their respective headers if `__STDC_WANT_MATH_SPEC_FUNCS__` is defined as a macro which expands to the integer constant 1 at the point in the source file where the appropriate header is included. [2)]

3 Functions declared in Clause 6 and its subclauses shall not be declared by their respective headers if `__STDC_WANT_MATH_SPEC_FUNCS__` is not defined as a macro at the point in the source file where the appropriate header is included.

4 Within a preprocessing translation unit, `__STDC_WANT_MATH_SPEC_FUNCS__` shall be defined identically for all inclusions of any headers from Clause 6. If `__STDC_WANT_MATH_SPEC_FUNCS__` is defined differently for any such inclusion, the implementation shall issue a diagnostic as if a preprocessor error directive was used.

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

1 Table 2 summarizes the functions that are added to header <math.h>. The detailed signatures are given in the synopsis.

2 Each of these functions is provided for arguments of type `float`, `double`, and `long double`. The signatures added to header <math.h> are:

```
// [6.2.1] 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);

// [6.2.2] 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);

// [6.2.3] beta function:
double       beta(double x, double y);
float        betaf(float x, float y);
long double  betal(long double x, long double y);
```

2) Future revisions of this International Standard may define meanings for other values of `__STDC_WANT_MATH_SPEC_FUNCS__`.

```
// [6.2.4] (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);

// [6.2.5] (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);

// [6.2.6] (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);

// [6.2.7] 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);

// [6.2.8] 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);

// [6.2.9] 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);

// [6.2.10] 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);

// [6.2.11] (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);

// [6.2.12] (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);

// [6.2.13] (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);
```

```
// [6.2.14] exponential integral:
double      expint(double x);
float       expintf(float x);
long double expintl(long double x);

// [6.2.15] Hermite polynomials:
double      hermite(unsigned n, double x);
float       hermitef(unsigned n, float x);
long double hermitel(unsigned n, long double x);

// [6.2.16] Laguerre polynomials:
double      laguerre(unsigned n, double x);
float       laguerref(unsigned n, float x);
long double laguerrel(unsigned n, long double x);

// [6.2.17] Legendre polynomials:
double      legendre(unsigned l, double x);
float       legendref(unsigned l, float x);
long double legendrel(unsigned l, long double x);

// [6.2.18] Riemann zeta function:
double      riemann_zeta(double);
float       riemann_zetaf(float);
long double riemann_zetal(long double);

// [6.2.19] 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);

// [6.2.20] 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);

// [6.2.21] 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);
```

Table 2: Additions to header <math.h> 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

3 Each of the functions declared above shall return a NaN (Not a Number) if any argument value is a NaN, but it shall not report a domain error. Otherwise, each of the functions declared above shall report a domain error for just those argument values for which:

— the function description's Returns clause explicitly specifies a domain, and those arguments fall outside the specified domain; or

— the corresponding mathematical function value has a non-zero imaginary component; or

— the corresponding mathematical function is not mathematically defined.[3)]

4 Unless otherwise specified, a function is defined for all finite values, for negative infinity, and for positive infinity.

6.2.1 associated Laguerre polynomials [num.sf.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 *Note:* The effect of calling each of these functions is implementation-defined if `n >= 128` or if `m >= 128`.

6.2.2 associated Legendre polynomials [num.sf.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);
```

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

1 *Effects:* These functions compute the associated Legendre functions of their respective arguments `l`, `m`, and `x`.

2 *Returns:* The `assoc_legendre` functions return

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

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

6.2.3 beta function [num.sf.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

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

6.2.4 (complete) elliptic integral of the first kind [num.sf.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 6.2.11.

6.2.5 (complete) elliptic integral of the second kind [num.sf.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 6.2.12.

6.2.6 (complete) elliptic integral of the third kind [num.sf.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

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

3 See 6.2.13.

6.2.7 regular modified cylindrical Bessel functions [num.sf.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

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

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

6.2.8 cylindrical Bessel functions (of the first kind) [num.sf.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

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

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

6.2.9 irregular modified cylindrical Bessel functions [num.sf.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 *Note:* The effect of calling each of these functions is implementation-defined if `nu >= 128`.

6.2.10 cylindrical Neumann functions [num.sf.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 *Note:* The effect of calling each of these functions is implementation-defined if `nu >= 128`.

4 See 6.2.8.

6.2.11 (incomplete) elliptic integral of the first kind [num.sf.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

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

6.2.12 (incomplete) elliptic integral of the second kind [num.sf.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

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

6.2.13 (incomplete) elliptic integral of the third kind [num.sf.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

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

6.2.14 exponential integral [num.sf.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

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

6.2.15 Hermite polynomials [num.sf.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

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

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

6.2.16 Laguerre polynomials [num.sf.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

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

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

6.2.17 Legendre polynomials [num.sf.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

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

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

6.2.18 Riemann zeta function [num.sf.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

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

6.2.19 spherical Bessel functions (of the first kind) [num.sf.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 *Note:* The effect of calling each of these functions is implementation-defined if `n >= 128`.

4 See 6.2.8.

6.2.20 spherical associated Legendre functions [num.sf.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 *Note:* The effect of calling each of these functions is implementation-defined if `l >= 128`.

4 See 6.2.8.

6.2.21 spherical Neumann functions [num.sf.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 *Note:* The effect of calling each of these functions is implementation-defined if `n >= 128`.

4 See 6.2.10.

6.3 Additions to header `<tgmath.h>` [sf.tgmath]

1 The header `<tgmath.h>` includes the header `<math.h>` and defines the type-generic macros shown in Table 3.

2 Of the functions added by this document to `<math.h>` without an `f` (float) or `l` (long double) suffix, several have one or more parameters whose corresponding real type is `double`. For each such function there is a corresponding type-generic macro. [4] The parameters whose corresponding real type is `double` in the function synopsis are generic parameters. Use of the macro invokes a function whose corresponding real type and type domain are determined by the arguments for the generic parameters. [5]

3 Use of the macro invokes a function whose generic parameters have the corresponding real type determined as follows:

— First, if any argument for generic parameters has type `long double`, the type determined is `long double`.

— Otherwise, if any argument for generic parameters has type `double` or is of integer type, the type determined is `double`.

— Otherwise, the type determined is `float`.

4 For each unsuffixed function added to `<math.h>` the corresponding type-generic macro has the same name as the function. These type-generic macros are shown in Table 3.

[4] Like other function-like macros in Standard libraries, each *type-generic macro* can be suppressed to make available the corresponding ordinary function.

[5] If the type of the argument is not compatible with the type of the parameter for the selected function, the behavior is undefined.

Table 3: Additions to header `<tgmath.h>` synopsis

Macros:

`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`

5 If all arguments for generic parameters are real, then use of the macro invokes a real function; otherwise, use of the macro results in undefined behavior.

Bibliography

[1] International Standards Organization: *Technical Report 1 on C++ Library Extensions*. International Standard ISO/IEC TR 19768:2006.

[2] International Standards Organization: *Quantities and units, Third edition*. International Standard ISO 31-11:1992. ISBN 92-67-10185-4.

[3] International Standards Organization: *Programming Languages – C, Second edition*. International Standard ISO/IEC 9899:1999.

Index