

ChorusOS man pages section 3M: Mathematical Libraries

Sun Microsystems, Inc. 901 San Antonio Road Palo Alto, CA 94303-4900 U.S.A.

> Part No: 806-3332 December 10, 1999

Copyright 1999 Sun Microsystems, Inc. 901 San Antonio Road, Palo Alto, California 94303-4900 U.S.A. All rights reserved.

This product or document is protected by copyright and distributed under licenses restricting its use, copying, distribution, and decompilation. No part of this product or document may be reproduced in any form by any means without prior written authorization of Sun and its licensors, if any. Third-party software, including font technology, is copyrighted and licensed from Sun suppliers.

Parts of the product may be derived from Berkeley BSD systems, licensed from the University of California. UNIX is a registered trademark in the U.S. and other countries, exclusively licensed through X/Open Company, Ltd.

Sun, Sun Microsystems, the Sun logo, docs.sun.com, AnswerBook, AnswerBook2, ChorusOS, and Solaris are trademarks, registered trademarks, or service marks of Sun Microsystems, Inc. in the U.S. and other countries. All SPARC trademarks are used under license and are trademarks or registered trademarks of SPARC International, Inc. in the U.S. and other countries. Products bearing SPARC trademarks are based upon an architecture developed by Sun Microsystems, Inc.

The OPEN LOOK and Sun[™] Graphical User Interface was developed by Sun Microsystems, Inc. for its users and licensees. Sun acknowledges the pioneering efforts of Xerox in researching and developing the concept of visual or graphical user interfaces for the computer industry. Sun holds a non-exclusive license from Xerox to the Xerox Graphical User Interface, which license also covers Sun's licensees who implement OPEN LOOK GUIs and otherwise comply with Sun's written license agreements.

RESTRICTED RIGHTS: Use, duplication, or disclosure by the U.S. Government is subject to restrictions of FAR 52.227-14(g)(2)(6/87) and FAR 52.227-19(6/87), or DFAR 252.227-7015(b)(6/95) and DFAR 227.7202-3(a).

DOCUMENTATION IS PROVIDED "AS IS" AND ALL EXPRESS OR IMPLIED CONDITIONS, REPRESENTATIONS AND WARRANTIES, INCLUDING ANY IMPLIED WARRANTY OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE OR NON-INFRINGEMENT, ARE DISCLAIMED, EXCEPT TO THE EXTENT THAT SUCH DISCLAIMERS ARE HELD TO BE LEGALLY INVALID.

Copyright 1999 Sun Microsystems, Inc. 901 San Antonio Road, Palo Alto, Californie 94303-4900 Etats-Unis. Tous droits réservés.

Ce produit ou document est protégé par un copyright et distribué avec des licences qui en restreignent l'utilisation, la copie, la distribution, et la décompilation. Aucune partie de ce produit ou document ne peut être reproduite sous aucune forme, par quelque moyen que ce soit, sans l'autorisation préalable et écrite de Sun et de ses bailleurs de licence, s'il y en a. Le logiciel détenu par des tiers, et qui comprend la technologie relative aux polices de caractères, est protégé par un copyright et licencié par des fournisseurs de Sun.

Des parties de ce produit pourront être dérivées du système Berkeley BSD licenciés par l'Université de Californie. UNIX est une marque déposée aux Etats-Unis et dans d'autres pays et licenciée exclusivement par X/Open Company, Ltd.

Sun, Sun Microsystems, le logo Sun, docs.sun.com, AnswerBook, AnswerBook2, ChorusOS, et Solaris sont des marques de fabrique ou des marques déposées, ou marques de service, de Sun Microsystems, Inc. aux Etats-Unis et dans d'autres pays. Toutes les marques SPARC sont utilisées sous licence et sont des marques de fabrique ou des marques déposées de SPARC International, Inc. aux Etats-Unis et dans d'autres pays. Les produits portant les marques SPARC sont basés sur une architecture développée par Sun Microsystems, Inc.

L'interface d'utilisation graphique OPEN LOOK et Sun™ a été développée par Sun Microsystems, Inc. pour ses utilisateurs et licenciés. Sun reconnaît les efforts de pionniers de Xerox pour la recherche et le développement du concept des interfaces d'utilisation visuelle ou graphique pour l'industrie de l'informatique. Sun détient une licence non exclusive de Xerox sur l'interface d'utilisation graphique Xerox, cette licence couvrant également les licenciés de Sun qui mettent en place l'interface d'utilisation graphique OPEN LOOK et qui en outre se conforment aux licences écrites de Sun.

CETTE PUBLICATION EST FOURNIE "EN L'ETAT" ET AUCUNE GARANTIE, EXPRESSE OU IMPLICITE, N'EST ACCORDEE, Y COMPRIS DES GARANTIES CONCERNANT LA VALEUR MARCHANDE, L'APTITUDE DE LA PUBLICATION A REPONDRE A UNE UTILISATION PARTICULIERE, OU LE FAIT QU'ELLE NE SOIT PAS CONTREFAISANTE DE PRODUIT DE TIERS. CE DENI DE GARANTIE NE S'APPLIQUERAIT PAS, DANS LA MESURE OU IL SERAIT TENU JURIDIQUEMENT NUL ET NON AVENU.





Contents

PREFACE 9 acos(3M) 15 acosh(3M) 16 asin(3M) 17 asinh(3M) 18 atan2(3M) 19 atan(3M) 20 atanh(3M) 21 hypot(3M) 22 cabs(3M) 22 sqrt(3M) 23 cbrt(3M) 23 ceil(3M) 24 ieee(3M) 25 copysign(3M) 25 drem(3M) 25 finite(3M) 25 logb(3M) 25 scalb(3M) 25

```
cos(3M) 26
```

cosh(3M) 27

ieee(3M) 28

copysign(3M) 28

drem(3M) 28

finite(3M) 28

logb(3M) 28

scalb(3M) 28

erf(3M) 29

erfc(3M) 29

erf(3M) 30

erfc(3M) 30

exp(3M) 31

expm1(3M) 31

log(3M) 31

log10(3M) 31

log1p(3M) 31

pow(3M) 31

exp(3M) 33

expm1(3M) 33

log(3M) 33

log10(3M) 33

log1p(3M) 33

pow(3M) 33

ieee(3M) 35

copysign(3M) 35

drem(3M) 35

finite(3M) 35

4 ChorusOS man pages section 3M: Mathematical Libraries ◆ December 10, 1999

- logb(3M) 35
- scalb(3M) 35
- floor(3M) 36
- fmod(3M) 37
- lgamma(3M) 38
- gamma(3M) 38
- hypot(3M) 40
- cabs(3M) 40
- ieee(3M) 41
- copysign(3M) 41
- drem(3M) 41
- finite(3M) 41
- logb(3M) 41
- scalb(3M) 41
- infnan(3M) 42
- j0(3M) 43
- j1(3M) 43
- jn(3M) 43
- y0(3M) 43
- y1(3M) 43
- yn(3M) 43
- j0(3M) 44
- j1(3M) 44
- jn(3M) 44
- y0(3M) 44
- y1(3M) 44
- yn(3M) 44
- j0(3M) 45

- j1(3M) 45
- jn(3M) 45
- y0(3M) 45
- y1(3M) 45
- yn(3M) 45
- lgamma(3M) 46
- gamma(3M) 46
- exp(3M) 48
- expm1(3M) 48
- log(3M) 48
- log10(3M) 48
- log1p(3M) 48
- pow(3M) 48
- exp(3M) 50
- expm1(3M) 50
- log(3M) 50
- log10(3M) 50
- log1p(3M) 50
- pow(3M) 50
- exp(3M) 52
- expm1(3M) 52
- log(3M) 52
- log10(3M) 52
- log1p(3M) 52
- pow(3M) 52
- ieee(3M) 54
- copysign(3M) 54
- drem(3M) 54

6

- finite(3M) 54
- logb(3M) 54
- scalb(3M) 54
- math(3M) 55
- exp(3M) 60
- expm1(3M) 60
- log(3M) 60
- log10(3M) 60
- log1p(3M) 60
- pow(3M) 60
- rint(3M) 62
- ieee(3M) 63
- copysign(3M) 63
- drem(3M) 63
- finite(3M) 63
- logb(3M) 63
- scalb(3M) 63
- sin(3M) 64
- sinh(3M) 65
- sqrt(3M) 66
- cbrt(3M) 66
- tan(3M) 67
- tanh(3M) 68
- j0(3M) 69
- j1(3M) 69
- jn(3M) 69
- y0(3M) 69
- y1(3M) 69

- yn(3M) 69
- j0(3M) 70
- j1(3M) 70
- jn(3M) 70
- y0(3M) 70
- y1(3M) 70
- yn(3M) 70
- j0(3M) 71
- j1(3M) 71
- jn(3M) 71
- y0(3M) 71
- y1(3M) 71
- yn(3M) 71
- Index 71

PREFACE

Overview

A man page is provided for both the naive user, and sophisticated user who is familiar with the $ChorusOS^{TM}$ operating system and is in need of on-line information. A man page is intended to answer concisely the question "What does it do?" The man pages in general comprise a reference manual. They are not intended to be a tutorial.

The following is a list of sections in the ChorusOS man pages and the information it references:

- Section 1CC: User Utilities; Host and Target Utilities
- Section 1M: System Management Utilities
- Section 2DL: System Calls; Data Link Services
- Section 2K: System Calls; Kernel Services
- Section 2MON: System Calls; Monitoring Services
- Section 2POSIX: System Calls; POSIX System Calls
- Section 2RESTART: System Calls; Hot Restart and Persistent Memory
- Section 2SEG: System Calls; Virtual Memory Segment Services
- Section 3FTPD: Libraries; FTP Daemon
- Section 3M: Libraries; Mathematical Libraries
- Section 3POSIX: Libraries; POSIX Library Functions
- Section 3RPC: Libraries; RPC Services
- Section 3STDC: Libraries; Standard C Library Functions
- Section 3TELD: Libraries; Telnet Services
- Section 4CC: Files

■ Section 5FEA: ChorusOS Features and APIs

■ Section 7P: Protocols

■ Section 7S: Services

Section 9DDI: Device Driver Interfaces
 Section 9DKI: Driver to Kernel Interface
 Section 9DRV: Driver Implementations

ChorusOS man pages are grouped in Reference Manuals, with one reference manual per section.

Below is a generic format for man pages. The man pages of each manual section generally follow this order, but include only needed headings. For example, if there are no bugs to report, there is no BUGS section. See the intro pages for more information and detail about each section, and man(1) for more information about man pages in general.

NAME

This section gives the names of the commands or functions documented, followed by a brief description of what they do.

SYNOPSIS

This section shows the syntax of commands or functions. When a command or file does not exist in the standard path, its full pathname is shown. Options and arguments are alphabetized, with single letter arguments first, and options with arguments next, unless a different argument order is required.

The following special characters are used in this section:

- [] The option or argument enclosed in these brackets is optional. If the brackets are omitted, the argument must be specified.
- . . . Ellipses. Several values may be provided for the previous argument, or the previous argument can be specified multiple times, for example, "filename . . . " .
- Separator. Only one of the arguments separated by this character can be specified at time.
- { } Braces. The options and/or arguments enclosed within braces are

interdependent, such that everything enclosed must be treated as a unit.

FEATURES This section provides the list of features which

offer an interface. An API may be associated with one or more system features. The interface will be available if one of the associated features

has been configured.

DESCRIPTION This section defines the functionality and

behavior of the service. Thus it describes concisely what the command does. It does not discuss OPTIONS or cite EXAMPLES.. Interactive commands, subcommands, requests, macros, functions and such, are described under USAGE.

OPTIONS This lists the command options with a concise

summary of what each option does. The options are listed literally and in the order they appear in the SYNOPSIS section. Possible arguments to options are discussed under the option, and where appropriate, default values are supplied.

OPERANDS This section lists the command operands and

describes how they affect the actions of the

command.

OUTPUT This section describes the output - standard

output, standard error, or output files - generated

by the command.

RETURN VALUES If the man page documents functions that

return values, this section lists these values and describes the conditions under which they are returned. If a function can return only constant values, such as 0 or -1, these values are listed in tagged paragraphs. Otherwise, a single paragraph describes the return values of each function. Functions declared void do not return values, so they are not discussed in RETURN

VALUES.

ERRORS On failure, most functions place an error code in

the global variable errno indicating why they failed. This section lists alphabetically all error codes a function can generate and describes the conditions that cause each error. When more than one condition can cause the same error, each condition is described in a separate paragraph

under the error code.

USAGE This section is provided as a guidance on use.

> This section lists special rules, features and commands that require in-depth explanations. The subsections listed below are used to explain

built-in functionality:

EXAMPLES

Commands
This section provides examples of usage or of Modifiers
how to use a command or function. Wherever variables possible a complete example including command line entry and machine response is shown line entry and machine response is shown.
Input Grammar
Whenever an example is given, the prompt is shown as example% or if the user must be superuser, example#. Examples are followed by explanations, variable substitution rules,

OPTIONS and USAGE sections.

ENVIRONMENT VARIABLES This section lists any environment variables that

the command or function affects, followed by a

or returned values. Most examples illustrate concepts from the SYNOPSIS, DESCRIPTION,

brief description of the effect.

EXIT STATUS This section lists the values the command returns

> to the calling program or shell and the conditions that cause these values to be returned. Usually, zero is returned for successful completion and values other than zero for various error

conditions.

FILES This section lists all filenames referred to by the

> man page, files of interest, and files created or required by commands. Each is followed by a

descriptive summary or explanation.

SEE ALSO This section lists references to other man

pages, in-house documentation and outside

publications.

DIAGNOSTICS This section lists diagnostic messages with a brief

explanation of the condition causing the error.

WARNINGS This section lists warnings about special

> conditions which could seriously affect your working conditions. This is not a list of

diagnostics.

This section lists additional information that **NOTES**

> does not belong anywhere else on the page. It takes the form of an aside to the user, covering points of special interest. Critical information is

never covered here.

This section describes known bugs and wherever possible, suggests workarounds.

Mathematical Library

Mathematical Library acos(3M)

NAME | acos – arc cosine function

SYNOPSIS #include <math.h>

double acos(double x);

DESCRIPTION The *acos* function computes the principal value of the arc cosine of x. A domain

error occurs for arguments not in the range [-1, +1]. For a discussion of errors

due to rounding off, see math(3M).

RETURN VALUES The *acos* function returns the arc cosine in the range [0, Pi] radians. If |x| > 1,

errno is set to EDOM and a system-dependent notification is performed.

ATTRIBUTES See attributes(5) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Evolving

SEE ALSO $\sin(3M)$, $\cos(3M)$, $\tan(3M)$, $a\sin(3M)$, $a\tan(3M)$,

STANDARDS The *acos* function conforms to ANSI-C.

NAME | acosh – inverse hyperbolic cosine function

SYNOPSIS #include <math.h>

double acosh(double x);

DESCRIPTION The *acosh* function computes the inverse hyperbolic cosine of the real argument

x. For a discussion of errors due to rounding off, see math(3M).

RETURN VALUES The *acosh* function returns the inverse hyperbolic cosine of *x*. If the argument is

less than one, *acosh* sets *errno* to EDOM and a system-dependent notification is

performed.

ATTRIBUTES See attributes(5) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Evolving

SEE ALSO asinh(3M), atanh(3M), exp(3M), infnan(3M), math(3M)

Mathematical Library asin(3M)

NAME | asin – arc sine function

SYNOPSIS #include <math.h>

double asin(double x);

DESCRIPTION The *asin* function computes the principal value of the arc sine of x. A domain

error occurs for arguments not in the range $\left[-1,+1\right]$. For a discussion of errors

due to rounding off, see math(3M).

RETURN VALUES The *asin* function returns the arc sine in the range [-Pi/2, +Pi/2] radians. If |x|

> 1, errno is set to EDOM and a system-dependent notification is performed.

ATTRIBUTES See attributes(5) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Evolving

SEE ALSO acos(3M), atan(3M), atan(3M), cos(3M), cosh(3M), sin(3M), sin(3M), tan(3M), tanh(3M), tanh(3M)

STANDARDS The *asin* function conforms to ANSI-C.

NAME | asinh – inverse hyperbolic sine function

SYNOPSIS #include <math.h>

double asinh(double x);

DESCRIPTION The *asinh* function computes the inverse hyperbolic sine of the real argument *x*.

For a discussion of errors due to rounding off, see math(3M).

RETURN VALUES The *asinh* function returns the inverse hyperbolic sine of *x*.

ATTRIBUTES See attributes(5) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Evolving

SEE ALSO acosh(3M), atanh(3M), exp(3M), infnan(3M), math(3M)

Mathematical Library atan2(3M)

NAME

atan2 – arc tangent function of two variables

SYNOPSIS

#include <math.h>

double atan2(double y, double x);

DESCRIPTION

The atan2 function computes the principal value of the arc tangent of y/x, using the signs of both arguments to determine the quadrant of the return value.

RETURN VALUES

The atan2 function, if successful, returns the arc tangent of y/x in radians. If both x and y are zero, *errno* is set to EDOM and a system-dependent notification is performed.

ATTRIBUTES

See attributes(5) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Evolving

SEE ALSO

acos(3M), asin(3M), atan(3M), cos(3M), cosh(3M), sin(3M), sin(3M), tan(3M), tan(3M), math(3M)

STANDARDS

The atan2 function conforms to ANSI-C.

NAME | atan – arc tangent function of one variable

SYNOPSIS #include <math.h> double atan(double *x*);

DESCRIPTION The *atan* function computes the principal value of the arc tangent of x. For a

discussion of errors due to rounding off, see math(3M).

RETURN VALUES The *atan* function returns the arc tangent in radians.

ATTRIBUTES See attributes(5) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Evolving

SEE ALSO $a\cos(3M)$, $a\sin(3M)$, $a\tan(3M)$, $\cos(3M)$, $\cos(3M)$, $\sin(3M)$, $\sin(3M)$, $\sin(3M)$,

tan(3M), tanh(3M), math(3M)

STANDARDS The *atan* function conforms to ANSI-C.

Mathematical Library atanh(3M)

NAME

atanh – inverse hyperbolic tangent function

SYNOPSIS

#include <math.h>
double atanh(double x);

The *atanh* function computes the inverse hyperbolic tangent of the real argument *x*. For a discussion of errors due to rounding off, see math(3M).

RETURN VALUES

DESCRIPTION

The *atanh* function returns the inverse hyperbolic tangent of *x*, if successful. If the argument has an absolute value greater than or equal to 1, *atanh* sets *errno* to EDOM and a system-dependent notification is performed.

ATTRIBUTES

See attributes(5) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Evolving

SEE ALSO

acosh(3M), asinh(3M), exp(3M), infnan(3M), math(3M)

NAME

hypot, cabs - euclidean distance and complex absolute value functions

SYNOPSIS

#include <math.h>

double hypot(double x, double y);

double cabs(struct {double x; double y;} z;);

DESCRIPTION

The *hypot* and *cabs* functions compute the square root of (x*x+y*y) in such a way that underflow will not occur, and overflow occurs only if the final result justifies it.

 $hypot (\infty, v) = hypot (v, \infty) = +Infinity$

for all values of v, including NaN.

ERRORS (due to Roundoff)

Less than 0.97 **ulp** s. Consequently, hypot(5.0, 12.0) = 13.0 exactly; in general,

hypot and cabs return an integer whenever an integer is expected.

ATTRIBUTES

See attributes(5) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Evolving

SEE ALSO

math(3M), sqrt(3M)

Mathematical Library cbrt(3M)

> **NAME** sqrt, cbrt – cube root and square root functions

SYNOPSIS #include <math.h>

double cbrt(double x);

double sqrt(double x);

DESCRIPTION The *cbrt* function computes the cube root of x.

The *sqrt* function computes the non-negative square root of x.

The *cbrt* function returns the requested cube root. The *sqrt* function returns the **RETURN VALUES**

requested square root unless an error occurs. An attempt to take the sqrt of a negative value of x causes an error; in this event, errno is set to EDOM and a

system-dependent notification is performed.

ERRORS (due to Roundoff)

The *cbrt* function is accurate to within 0.7 ulps.

On a machine that conforms to IEEE 754 sqrt is correctly rounded in accordance

with the rounding mode in force; the error is less than half a ulp in the default mode (round-to-nearest). A ulp is one Unit in the Last Place carried.

ATTRIBUTES

See attributes(5) for descriptions of the following attributes:

	ATTRIBUTE TYPE	ATTRIBUTE VALUE
Inter	face Stability	Evolving

SEE ALSO

math(3M), infnan(3M)

STANDARDS

The sqrt function conforms to ANSI-C.

NAME | ceil – smallest integral value not less than x

SYNOPSIS #include <math.h> double ceil(double x);

DESCRIPTION The *ceil* function computes the smallest integral value not less than *x*.

RETURN VALUES The *ceil* function returns the smallest integral value expressed as a double.

ATTRIBUTES See attributes(5) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Evolving

SEE ALSO floor(3M), rint(3M), ieee(3M), math(3M) abs(3STDC) fabs(3STDC)

STANDARDS The *ceil* function conforms to ANSI-C.

Mathematical Library copysign(3M)

NAME

ieee, copysign, drem, finite, logb, scalb - IEEE 754 floating point support

SYNOPSIS

#include <math.h>

double copysign(double x, double y);

double drem(double x, double y);

int finite(double x);

double logb(double x);

double scalb(double x, int n);

DESCRIPTION

These functions are required for the ${\tt IEEE}$ 754 standard for floating-point arithmetic.

The *copysign* () function returns x with its sign changed to that of y.

The *drem ()* function returns the remainder $r := x - n^*y$ where n is the integer closest to the exact value of x/y. If |n - x/y| = 1/2, n is even. Consequently, the remainder is computed exactly and |r| <= |y| / 2. Note that *drem* (x, 0) is exceptional.

The *finite ()* function returns true (1) if the argument *x* is neither inifinity nor *NaN* value. Otherwise it returns false (0).

The *logb* () function computes the exponent of x, which is the integral part of log2 | x |, as a signed floating point value for non-zero x.

The *scalb* () function returns $x * (2^{**}n)$ computed, for integer n, without first computing $2^{**}n$.

RETURN VALUES

The IEEE 754 standard defines drem(x, 0) and drem(infinity , y) to be invalid operations that produce a **NaN** .

IEEE 754 defines logb (\pm infinity) = + infinity, and logb (0) = -HUGE_VAL, and requires the latter to signal Division–by–Zero. Upon successful completion, logb() returns the exponent of x.

ATTRIBUTES

See attributes(5) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Evolving

SEE ALSO

floor(3M), math(3M), infnan(3M)

NAME | cos – cosine function

SYNOPSIS #include <math.h>

double cos(double x);

DESCRIPTION The *cos* function computes the cosine of *x* (measured in radians). A large

magnitude argument may yield a result with little or no significance. For a

discussion of errors due to rounding off, see math(3M).

RETURN VALUES The *cos* function returns the cosine value.

ATTRIBUTES See attributes(5) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Evolving

SEE ALSO sin(3M), tan(3M), asin(3M), acos(3M), atan(3M), atan(3M), sinh(3M),

cosh(3M), tanh(3M), math(3M)

STANDARDS The *cos* function conforms to ANSI-C.

Mathematical Library cosh(3M)

NAME | cosh – hyperbolic cosine function

SYNOPSIS #include <math.h>

double cosh(double x);

DESCRIPTION The *cosh* function computes the hyperbolic cosine of x.

RETURN VALUES The *cosh* function returns the hyperbolic cosine, unless the magnitude of *x*

is too large; in this event, errno is set to ERANGE and a system-dependent

notification is performed.

ATTRIBUTES See attributes(5) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Evolving

SEE ALSO acos(3M), asin(3M), atan(3M), atan2(3M), cos(3M), sin(3M), sin(3M),

tan(3M), tanh(3M), math(3M)

STANDARDS The *cosh* function conforms to ANSI-C.

NAME

ieee, copysign, drem, finite, logb, scalb - IEEE 754 floating point support

SYNOPSIS

#include <math.h>

double copysign(double x, double y);

double drem(double x, double y);

int finite(double x);

double logb(double x);

double scalb(double x, int n);

DESCRIPTION

These functions are required for the ${\tt IEEE}$ 754 standard for floating-point arithmetic.

The *copysign* () function returns *x* with its sign changed to that of *y* .

The *drem ()* function returns the remainder $r := x - n^*y$ where n is the integer closest to the exact value of x/y. If |n-x/y| = 1/2, n is even. Consequently, the remainder is computed exactly and |r| <= |y| / 2. Note that *drem* (x, 0) is exceptional.

The *finite* () function returns true (1) if the argument *x* is neither inifinity nor *NaN* value. Otherwise it returns false (0).

The *logb* () function computes the exponent of x, which is the integral part of log2 | x |, as a signed floating point value for non-zero x.

The *scalb* () function returns $x * (2^{**}n)$ computed, for integer n, without first computing $2^{**}n$.

RETURN VALUES

The IEEE 754 standard defines drem(x, 0) and drem(infinity, y) to be invalid operations that produce a **NaN**.

IEEE 754 defines logb (\pm infinity) = + infinity, and logb (0) = $-HUGE_VAL$, and requires the latter to signal Division–by–Zero. Upon successful completion, logb() returns the exponent of x.

ATTRIBUTES

See attributes(5) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Evolving

SEE ALSO

floor(3M), math(3M), infnan(3M)

28

ChorusOS 4.0

Last modified December 1999

Mathematical Library erf(3M)

> NAME erf, erfc – error function operators

SYNOPSIS #include <math.h>

double erf(double x);

double erfc(double x);

DESCRIPTION These functions calculate the error function of x.

The *erf* function calculates the error function of x; where

erf(x) = 2/sqrt(pi)*integral from 0 to x of exp(-t*t) dt.

The *erfc* function calculates the complementary error function of x; that is, *erfc* subtracts the result of the error function erf(x) from 1.0. This is useful when x is a

large value, as decimal places can be lost.

ATTRIBUTES See attributes(5) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Evolving

SEE ALSO

math(3M)

NAME | erf, erfc – error function operators

SYNOPSIS #include <math.h> double erf(double *x*);

double erfc(double x);

DESCRIPTION These functions calculate the error function of x.

The *erf* function calculates the error function of x; where

erf(x) = 2/sqrt(pi)*integral from 0 to x of exp(-t*t) dt.

The *erfc* function calculates the complementary error function of x; that is, *erfc* subtracts the result of the error function erf(x) from 1.0. This is useful when x is a

large value, as decimal places can be lost.

ATTRIBUTES See attributes(5) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Evolving

SEE ALSO math(3M)

Mathematical Library exp(3M)

NAME

exp, expm1, log, log10, log1p, pow - exponential, logarithm, power functions

SYNOPSIS

#include <math.h>
double exp(double x);

double CLLP (double 12),

double expm1(double x);
double log(double x);

double log10(double x);

double log1p(double x);

double pow(double x, double y);

DESCRIPTION

The exp function computes the exponential value of the given argument x.

The *expm1* function computes the value $\exp(x)-1$ accurately even for extremely small values of x.

The *log* function computes the natural logarithm of the argument x.

The *log10* function computes the logarithm of argument *x* to base 10.

The log1p function computes the value of log(1+x) accurately even for extremely small values of x.

The pow computes the value of to the exponent

ERRORS (due to Roundoff)

The exp(x), log(x), expm1(x) and log1p(x) functions are accurate to within a ulp, and log10(x) to within approximately 2 ulps. A ulp is one Unit in the Last Place. The error in pow(x, y) is below about 2 ulps when its magnitude is moderate, but increases as pow(x, y) approaches the over/underflow thresholds. Almost as many bits as are occupied by the floating-point format's exponent field could be lost; that is 11 bits for IEEE 754 Double. The worst errors observed during testing have been under 300 ulps for IEEE 754 Double. Moderate values of pow are sufficiently accurate that pow(integer, integer) is precise until it is greater than $2^{**}53$ for IEEE 754.

RETURN VALUES

These functions will return the appropriate computation unless an error occurs or an argument is out of range. The functions exp, expm1 and pow detect if the computed value will overflow, set errno to ERANGE and a system-dependent notification is performed. The pow(x, y) function checks whether x < 0 and y is not an integer; if this is true, errno is set to EDOM and a system-dependent notification is performed. errno is set to EDOM and a system-dependent notification is performed by log, unless x > 0, by log1p unless x > -1.

NOTES

The function pow(x, 0) returns $x^{**}0 = 1$ for all x including x = 0, and NaN.

ATTRIBUTES

See attributes(5) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Evolving

SEE ALSO

math(3M), infnan(3M)

Mathematical Library expm1(3M)

NAME

exp, expm1, log, log10, log1p, pow - exponential, logarithm, power functions

SYNOPSIS

#include <math.h>
double exp(double x);

double expm1(double x);

double log(double x);

double log10(double x);

double log1p(double x);

double pow(double x, double y);

DESCRIPTION

The exp function computes the exponential value of the given argument x.

The *expm1* function computes the value $\exp(x)-1$ accurately even for extremely small values of x.

The *log* function computes the natural logarithm of the argument x.

The *log10* function computes the logarithm of argument *x* to base 10.

The log1p function computes the value of log(1+x) accurately even for extremely small values of x.

The pow computes the value of to the exponent

ERRORS (due to Roundoff)

The exp(x), log(x), expm1(x) and log1p(x) functions are accurate to within a ulp, and log10(x) to within approximately 2 ulp s. A ulp is one Unit in the Last Place. The error in pow(x, y) is below about 2 ulp s when its magnitude is moderate, but increases as pow(x, y) approaches the over/underflow thresholds. Almost as many bits as are occupied by the floating-point format's exponent field could be lost; that is 11 bits for IEEE 754 Double. The worst errors observed during testing have been under 300 ulp s for IEEE 754 Double. Moderate values of pow are sufficiently accurate that pow(integer, integer) is precise until it is greater than 2^{**53} for IEEE 754.

RETURN VALUES

These functions will return the appropriate computation unless an error occurs or an argument is out of range. The functions exp, expm1 and pow detect if the computed value will overflow, set errno to ERANGE and a system-dependent notification is performed. The pow(x, y) function checks whether x < 0 and y is not an integer; if this is true, errno is set to EDOM and a system-dependent notification is performed. errno is set to EDOM and a system-dependent notification is performed by log, unless x > 0, by log1p unless x > -1.

NOTES

The function pow(x, 0) returns $x^{**}0 = 1$ for all x including x = 0, and NaN.

ATTRIBUTES

See attributes(5) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Evolving

SEE ALSO

math(3M), infnan(3M)

Mathematical Library finite(3M)

NAME

ieee, copysign, drem, finite, logb, scalb – IEEE 754 floating point support

SYNOPSIS

#include <math.h>

double copysign(double x, double y);

double drem(double x, double y);

int finite(double x);

double logb(double x);

double scalb(double x, int n);

DESCRIPTION

These functions are required for the ${\tt IEEE}$ 754 standard for floating-point arithmetic.

The *copysign* () function returns x with its sign changed to that of y.

The *drem ()* function returns the remainder $r := x - n^*y$ where n is the integer closest to the exact value of x/y. If |n - x/y| = 1/2, n is even. Consequently, the remainder is computed exactly and |r| <= |y| / 2. Note that *drem* (x, 0) is exceptional.

The *finite ()* function returns true (1) if the argument *x* is neither inifinity nor *NaN* value. Otherwise it returns false (0).

The *logb* () function computes the exponent of x, which is the integral part of log2 | x |, as a signed floating point value for non-zero x.

The *scalb* () function returns $x * (2^{**}n)$ computed, for integer n, without first computing $2^{**}n$.

RETURN VALUES

The IEEE 754 standard defines drem(x, 0) and drem(infinity , y) to be invalid operations that produce a **NaN** .

IEEE 754 defines logb (\pm infinity) = + infinity, and logb (0) = -HUGE_VAL, and requires the latter to signal Division–by–Zero. Upon successful completion, logb() returns the exponent of x.

ATTRIBUTES

See attributes(5) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Evolving

SEE ALSO

floor(3M), math(3M), infnan(3M)

NAME | floor – largest integral value not greater than x

SYNOPSIS #include <math.h>

double floor(double x);

DESCRIPTION The *floor* function computes the largest integral value not greater than *x*.

RETURN VALUES The *floor* function returns the largest integral value expressed as a double.

ATTRIBUTES See attributes(5) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Evolving

SEE ALSO

, ieee(3M), rint(3M), math(3M), abs(3STDC), fabs(3STDC)

STANDARDS

The *floor* function conforms to ANSI-C.

Mathematical Library fmod(3M)

NAME | fmod – floating-point remainder function

SYNOPSIS #include <math.h>

double fmod(double x, double y);

DESCRIPTION The *fmod* function computes the floating-point remainder of x/y.

RETURN VALUES The *fmod* function returns the value x - i * y, for the integer i. If y is non-zero, the

result has the same sign as *x* and a magnitude less than the magnitude of *y*. If *y* is zero, *errno* is set to EDOM and a system-dependent notification is performed.

ATTRIBUTES See attributes(5) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Evolving

SEE ALSO

math(3M)

STANDARDS

The fmod function conforms to ANSI-C.

lgamma, gamma - log gamma function, gamma function

SYNOPSIS

#include <math.h>; extern int signgam

double lgamma(double x);

double gamma(double x);

DESCRIPTION

lgamma(x) returns:

 $\ln | | \sim (x) |$

where

 $|\sim(x)| = \text{integral from } 0 \text{ to } +\infty \text{ of pow}(t, x-1) \exp(-t) \text{ dt for } x>0$

and

 $| \sim (x) = Pi/(| \sim (1-x)sin(pi^*x))$ for x < 0

The external integer *signgam* returns the sign of $|\sim(x)$.

gamma(x) returns $| \sim (x)$, with no effect on signgam.

IDIOSYNCRASIES

Do not use the expression

signgam*exp(lgamma(x)) to compute $g := | \sim (x)$.

Instead, use a program like this (in C):

lg = lgamma(x); g = signgam*exp(lg);

signgam will only be correct after lgamma has returned.

For arguments within its range, *gamma* is preferable, as for positive arguments it is accurate to within one unit in the last place. Exponentiation of *lgamma* will lose up to 10 significant bits.

Note - The *Igamma* function is not thread safe.

RETURN VALUES

The *gamma* and *lgamma* functions return appropriate values unless an argument is out of range. Overflow will occur for sufficiently large positive values, and non-positive integers. In this case, infinity is returned, *errno* is set to ERANGE , and a system-dependent notification is performed. For large non-integer negative values, *gamma* will underflow.

ATTRIBUTES

See attributes(5) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Evolving

SEE ALSO

math(3M), infnan(3M)

38

ChorusOS 4.0

Mathematical Library gamma(3M)

NOTES Due to *signgam* being a global variable, this routine is not reentrant.

hypot, cabs - euclidean distance and complex absolute value functions

SYNOPSIS

#include <math.h>

double hypot(double x, double y);

double $cabs(struct \{double x; double y;\} z;);$

DESCRIPTION

The *hypot* and *cabs* functions compute the square root of (x*x+y*y) in such a way that underflow will not occur, and overflow occurs only if the final result justifies it.

14.50.11.00

hypot $(\infty, \mathbf{v}) = hypot(\mathbf{v}, \infty) = +Infinity$ for all values of \mathbf{v} , including **NaN**.

ERRORS (due to Roundoff)

Less than 0.97 **ulp** s. Consequently, hypot(5.0, 12.0) = 13.0 exactly; in general, hypot and cabs return an integer whenever an integer is expected.

ATTRIBUTES

See attributes(5) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Evolving

SEE ALSO

math(3M), sqrt(3M)

40 ChorusOS 4.0

Mathematical Library ieee(3M)

NAME

ieee, copysign, drem, finite, logb, scalb - IEEE 754 floating point support

SYNOPSIS

#include <math.h>

double copysign(double x, double y);

double drem(double x, double y);

int finite(double x);

double logb(double x);

double scalb(double x, int n);

DESCRIPTION

These functions are required for the ${\tt IEEE}$ 754 standard for floating-point arithmetic.

The *copysign* () function returns x with its sign changed to that of y.

The *drem ()* function returns the remainder $r := x - n^*y$ where n is the integer closest to the exact value of x/y. If |n - x/y| = 1/2, n is even. Consequently, the remainder is computed exactly and |r| <= |y| / 2. Note that *drem* (x, 0) is exceptional.

The *finite ()* function returns true (1) if the argument *x* is neither inifinity nor *NaN* value. Otherwise it returns false (0).

The *logb* () function computes the exponent of x, which is the integral part of log2 | x |, as a signed floating point value for non-zero x.

The *scalb* () function returns x^* (2**n) computed, for integer n, without first computing 2**n.

RETURN VALUES

The IEEE 754 standard defines drem(x, 0) and drem(infinity , y) to be invalid operations that produce a **NaN** .

IEEE 754 defines logb (\pm infinity) = + infinity, and logb (0) = -HUGE_VAL, and requires the latter to signal Division–by–Zero. Upon successful completion, logb() returns the exponent of x.

ATTRIBUTES

See attributes(5) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Evolving

SEE ALSO

floor(3M), math(3M), infnan(3M)

infnan – signals invalid floating point operations

SYNOPSIS

#include <math.h>
double infnan(int iarg);

DESCRIPTION

Invalid, Overflow and Divide-by-Zero events are notified to the application by calls to *infnan* in appropriate places in *libm*. As exception-handling depends upon the operating system, *infnan* does not necessarily alter the current flow of control. Users of libm can design their own *infnan*.

Whenever an elementary function code in libm runs into an exceptional situation, or has to return an invalid result, it calls <code>infnan(iarg)</code> with an appropriate value of <code>iarg</code> (<code>ERANGE</code> or <code>EDOM</code>). The <code>infnan</code> function assigns the corresponding value to <code>errno</code> and triggers whatever exception mechanism is available. If given back control, it returns a non-finite value, which allows computation to resume, and prompts the user to consult the <code>errno</code> file.

ERANGE and EDOM are defined in errno.h.

ATTRIBUTES

See attributes(5) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Evolving

SEE ALSO

math(3M)

42 ChorusOS 4.0 Last modified December 1999

Mathematical Library j0(3M)

NAME

j0, j1, jn, y0, y1, yn - Bessel functions of the first and second kind

SYNOPSIS

#include <math.h>
double jo(double x);

double j1(double x);

double jn(int n, double x);

double y0(double x);

double **y1**(double *x*);

double **yn**(int *n*, double *x*);

DESCRIPTION

The j0 and j1 functions compute the Bessel function of the first type of the order 0 and the order 1, respectively, for the real value x. The jn function computes the Bessel function of the first type of the integer order n for the real value x.

RETURN VALUES

If these functions are successful, the computed value is returned. Upon successful completion, j00, j10 and jn0 return the relevant Bessel value of x of the first type.

Upon successful completion, y00, y10 and yn0 return the relevant Bessel value of x of the second type. If the x argument of y00, y10 or yn0 is negative, -huge_val is returned and errno is set errno is set to EDOM. If x is 0.0, -huge_val is returned and errno is set to ERANGE. If the correct result would cause overflow, -huge_val or huge_val is returned and errno is set to ERANGE

ATTRIBUTES

See attributes(5) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Evolving

SEE ALSO

math(3M), infnan(3M)

j0, j1, jn, y0, y1, yn - Bessel functions of the first and second kind

SYNOPSIS

#include <math.h>
double j0(double x);

double j1(double x);

double jn(int *n*, double *x*);

double y0(double x);

double **y1**(double *x*);

double yn(int n, double x);

DESCRIPTION

The j0 and j1 functions compute the Bessel function of the first type of the order 0 and the order 1, respectively, for the real value x. The jn function computes the Bessel function of the first type of the integer order n for the real value x.

RETURN VALUES

If these functions are successful, the computed value is returned. Upon successful completion, j00, j10 and jn0 return the relevant Bessel value of x of the first type.

Upon successful completion, y0(), y1() and yn() return the relevant Bessel value of x of the second type. If the x argument of y0(), y1() or yn() is negative, -huge_val is returned and errno is set to EDOM. If x is 0.0, -huge_val is returned and errno is set to ERANGE. If the correct result would cause overflow, -huge_val or huge_val is returned and errno is set to ERANGE

ATTRIBUTES

See attributes(5) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Evolving

SEE ALSO

math(3M), infnan(3M)

44

ChorusOS 4.0

Mathematical Library jn(3M)

NAME

j0, j1, jn, y0, y1, yn – Bessel functions of the first and second kind

SYNOPSIS

#include <math.h>
double j0(double x);

double j1(double x);

double jn(int n, double x);

double y0(double x);

double **y1**(double *x*);

double **yn**(int *n*, double *x*);

DESCRIPTION

The j0 and j1 functions compute the Bessel function of the first type of the order 0 and the order 1, respectively, for the real value x. The jn function computes the Bessel function of the first type of the integer order n for the real value x.

RETURN VALUES

If these functions are successful, the computed value is returned. Upon successful completion, j00, j10 and jn0 return the relevant Bessel value of x of the first type.

Upon successful completion, y0(), y1() and yn() return the relevant Bessel value of x of the second type. If the x argument of y0(), y1() or yn() is negative, -huge_val is returned and errno is set to EDOM. If x is 0.0, -huge_val is returned and errno is set to ERANGE. If the correct result would cause overflow, -huge_val or huge_val is returned and errno is set to ERANGE

ATTRIBUTES

See attributes(5) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Evolving

SEE ALSO

math(3M), infnan(3M)

lgamma, gamma – log gamma function, gamma function

SYNOPSIS

#include <math.h>; extern int signgam

double lgamma(double x);

double gamma(double x);

DESCRIPTION

lgamma(x) returns:

 $\ln | | \sim (x) |$

where

 $|\sim(x)| = \text{integral from } 0 \text{ to } +\infty \text{ of pow(t, } x-1) \exp(-t) \text{ dt for } x>0$

and

 $| \sim (x) = Pi/(| \sim (1-x)sin(pi^*x))$ for x < 0

The external integer *signgam* returns the sign of $|\sim(x)$.

gamma(x) returns $|\sim(x)$, with no effect on signgam.

IDIOSYNCRASIES

Do not use the expression

signgam*exp(lgamma(x)) to compute $g := | \sim (x)$.

Instead, use a program like this (in C):

lg = lgamma(x); g = signgam*exp(lg);

signgam will only be correct after lgamma has returned.

For arguments within its range, *gamma* is preferable, as for positive arguments it is accurate to within one unit in the last place. Exponentiation of *lgamma* will lose up to 10 significant bits.

Note - The *lgamma* function is not thread safe.

RETURN VALUES

The *gamma* and *lgamma* functions return appropriate values unless an argument is out of range. Overflow will occur for sufficiently large positive values, and non-positive integers. In this case, infinity is returned, *errno* is set to ERANGE , and a system-dependent notification is performed. For large non-integer negative values, *gamma* will underflow.

ATTRIBUTES

See attributes(5) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Evolving

SEE ALSO

math(3M), infnan(3M)

46

ChorusOS 4.0

Mathematical Library lgamma(3M)

NOTES Due to *signgam* being a global variable, this routine is not reentrant.

47

log10(3M) Mathematical Library

NAME

exp, expm1, log, log10, log1p, pow - exponential, logarithm, power functions

SYNOPSIS

#include <math.h>
double exp(double x);

double expm1(double x);

double log(double x);

double log10(double x);

double log1p(double x);

double pow(double x, double y);

DESCRIPTION

The exp function computes the exponential value of the given argument x.

The *expm1* function computes the value $\exp(x)-1$ accurately even for extremely small values of x.

The *log* function computes the natural logarithm of the argument x.

The *log10* function computes the logarithm of argument *x* to base 10.

The log1p function computes the value of log(1+x) accurately even for extremely small values of x .

The *pow* computes the value of to the exponent

ERRORS (due to Roundoff)

The exp(x), log(x), expm1(x) and log1p(x) functions are accurate to within a ulp, and log10(x) to within approximately 2 ulp s. A ulp is one Unit in the Last Place. The error in pow(x, y) is below about 2 ulp s when its magnitude is moderate, but increases as pow(x, y) approaches the over/underflow thresholds. Almost as many bits as are occupied by the floating-point format's exponent field could be lost; that is 11 bits for IEEE 754 Double. The worst errors observed during testing have been under 300 ulp s for IEEE 754 Double. Moderate values of pow are sufficiently accurate that pow(integer, integer) is precise until it is greater than $2^{**}53$ for IEEE 754.

RETURN VALUES

These functions will return the appropriate computation unless an error occurs or an argument is out of range. The functions exp, expm1 and pow detect if the computed value will overflow, set errno to ERANGE and a system-dependent notification is performed. The pow(x, y) function checks whether x < 0 and y is not an integer; if this is true, errno is set to EDOM and a system-dependent notification is performed. errno is set to EDOM and a system-dependent notification is performed by log, unless x > 0, by log1p unless x > -1.

NOTES

The function pow(x, 0) returns $x^{**}0 = 1$ for all x including x = 0, , and NaN.

ATTRIBUTES

See attributes(5) for descriptions of the following attributes:

48

ChorusOS 4.0

Mathematical Library log10(3M)

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Evolving

SEE ALSO

 ${\tt math}(3M)$, ${\tt infnan}(3M)$

log1p(3M) Mathematical Library

NAME

exp, expm1, log, log10, log1p, pow - exponential, logarithm, power functions

SYNOPSIS

#include <math.h>
double exp(double x);

double expm1(double x);

double log(double x);

double log10(double x);

double log1p(double x);

double pow(double x, double y);

DESCRIPTION

The exp function computes the exponential value of the given argument x.

The *expm1* function computes the value $\exp(x)-1$ accurately even for extremely small values of x.

The *log* function computes the natural logarithm of the argument x.

The *log10* function computes the logarithm of argument *x* to base 10.

The log1p function computes the value of log(1+x) accurately even for extremely small values of x .

The *pow* computes the value of to the exponent

ERRORS (due to Roundoff)

The exp(x), log(x), expm1(x) and log1p(x) functions are accurate to within a ulp, and log10(x) to within approximately 2 ulp s. A ulp is one Unit in the Last Place. The error in pow(x, y) is below about 2 ulp s when its magnitude is moderate, but increases as pow(x, y) approaches the over/underflow thresholds. Almost as many bits as are occupied by the floating-point format's exponent field could be lost; that is 11 bits for IEEE 754 Double. The worst errors observed during testing have been under 300 ulp s for IEEE 754 Double. Moderate values of pow are sufficiently accurate that pow(integer, integer) is precise until it is greater than $2^{**}53$ for IEEE 754.

RETURN VALUES

These functions will return the appropriate computation unless an error occurs or an argument is out of range. The functions exp, expm1 and pow detect if the computed value will overflow, set errno to ERANGE and a system-dependent notification is performed. The pow(x, y) function checks whether x < 0 and y is not an integer; if this is true, errno is set to EDOM and a system-dependent notification is performed. errno is set to EDOM and a system-dependent notification is performed by log, unless x > 0, by log1p unless x > -1.

NOTES

The function pow(x, 0) returns $x^{**}0 = 1$ for all x including x = 0, , and NaN.

ATTRIBUTES

See attributes(5) for descriptions of the following attributes:

50

ChorusOS 4.0

Mathematical Library log1p(3M)

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Evolving

SEE ALSO

 ${\tt math}(3M)$, ${\tt infnan}(3M)$

log(3M) Mathematical Library

NAME

exp, expm1, log, log10, log1p, pow - exponential, logarithm, power functions

SYNOPSIS

#include <math.h>
double exp(double x);

double expm1(double x);

double log(double x);

double log10(double x);

double log1p(double x);

double pow(double *x*, double *y*);

DESCRIPTION

The exp function computes the exponential value of the given argument x.

The *expm1* function computes the value $\exp(x)-1$ accurately even for extremely small values of x.

The *log* function computes the natural logarithm of the argument x.

The *log10* function computes the logarithm of argument *x* to base 10.

The log1p function computes the value of log(1+x) accurately even for extremely small values of x .

The *pow* computes the value of to the exponent

ERRORS (due to Roundoff)

The exp(x), log(x), expm1(x) and log1p(x) functions are accurate to within a ulp, and log10(x) to within approximately 2 ulp s. A ulp is one Unit in the Last Place. The error in pow(x, y) is below about 2 ulp s when its magnitude is moderate, but increases as pow(x, y) approaches the over/underflow thresholds. Almost as many bits as are occupied by the floating-point format's exponent field could be lost; that is 11 bits for IEEE 754 Double. The worst errors observed during testing have been under 300 ulp s for IEEE 754 Double. Moderate values of pow are sufficiently accurate that pow(integer, integer) is precise until it is greater than $2^{**}53$ for IEEE 754.

RETURN VALUES

These functions will return the appropriate computation unless an error occurs or an argument is out of range. The functions exp, expm1 and pow detect if the computed value will overflow, set errno to ERANGE and a system-dependent notification is performed. The pow(x, y) function checks whether x < 0 and y is not an integer; if this is true, errno is set to EDOM and a system-dependent notification is performed. errno is set to EDOM and a system-dependent notification is performed by log, unless x > 0, by log1p unless x > -1.

NOTES

The function pow(x, 0) returns $x^{**}0 = 1$ for all x including x = 0, , and NaN.

ATTRIBUTES

See attributes(5) for descriptions of the following attributes:

52

ChorusOS 4.0

Mathematical Library log(3M)

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Evolving

SEE ALSO

math(3M), infnan(3M)

ieee, copysign, drem, finite, logb, scalb - IEEE 754 floating point support

SYNOPSIS

#include <math.h>

double copysign(double x, double y);

double drem(double x, double y);

int finite(double x);

double logb(double x);

double scalb(double x, int n);

DESCRIPTION

These functions are required for the ${\tt IEEE}$ 754 standard for floating–point arithmetic.

The *copysign* () function returns *x* with its sign changed to that of *y* .

The *drem* () function returns the remainder $r := x - n^*y$ where n is the integer closest to the exact value of x/y. If |n - x/y| = 1/2, n is even. Consequently, the remainder is computed exactly and |r| <= |y| / 2. Note that *drem* (x, 0) is exceptional.

The *finite ()* function returns true (1) if the argument *x* is neither inifinity nor *NaN* value. Otherwise it returns false (0).

The *logb* () function computes the exponent of x, which is the integral part of log2 | x |, as a signed floating point value for non-zero x.

The *scalb* () function returns $x * (2^{**}n)$ computed, for integer n, without first computing $2^{**}n$.

RETURN VALUES

The IEEE 754 standard defines drem(x, 0) and drem(infinity , y) to be invalid operations that produce a **NaN** .

IEEE 754 defines logb (\pm infinity) = + infinity, and logb (0) = -HUGE_VAL, and requires the latter to signal Division–by–Zero. Upon successful completion, logb() returns the exponent of x.

ATTRIBUTES

See attributes(5) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Evolving

SEE ALSO

floor(3M), math(3M), infnan(3M)

54

ChorusOS 4.0

Mathematical Library math(3M)

NAME

DESCRIPTION

math – introduction to mathematical library functions

These functions constitute the C math library, *libm*. The link editor searches this library under the -lm option. Declarations for these functions may be obtained from the include file <math.h>. References of the form name(3M) refer to pages in this section of this document.

LIST OF FUNCTIONS

Name	Appears on Pag	ge Description	Error Bound (ULPs)
acos	acos.3	inverse trigonometric functi	on 3
acosh	acosh.3	inverse hyperbolic function	3
asin	asin.3	inverse trigonometric functi	
asinh	asin.3	inverse hyperbolic function	3
atan	sin.3	inverse trigonometric functi	
atanh	atanh.3	inverse hyperbolic function	3
atann	atan1.3	inverse trigonometric functi	
cabs	hypot.3	complex absolute value	1
cbrt	sqrt.3	cube root	1
ceil	ceil.3	integer no less than	0
copysign	ieee.3	copy sign bit	0
cos	cos.3	trigonometric function	1
cosh	cosh.3	3	3
drem	ieee.3	hyperbolic function remainder	0
			· ·
erf	erf.3	error function	???
erfc	erf.3	complementary error function	
exp	exp.3	exponential	1
expm1	exp.3	$\exp(x) - 1$	
finite	ieee.3	Is the value a valid finite	
floor	floor.3	integer no greater than	0
fmod	fmod.3	floating-point remainder	???
gamma	lgamma.3	gamma function	4
hypot	hypot.3	Euclidean distance	1
infnan	infnan.3	signals exceptions	
j0	j0.3	bessel function	???
j1	j0.3	bessel function	???
jn	j0.3	bessel function	???
lgamma	lgamma.3		2 for positive arguments
log	exp.3	natural logarithm	1
logb	ieee.3	exponent extraction	0
log10	exp.3	logarithm to base 10	3
log1p	exp.3	log(1+x) 1	
pow	exp.3	exponential x**y	60-500
rint	rint.3	round to nearest integer	0
scalb	ieee.3	exponent adjustment	0
sin	sin.3	trigonometric function	1
sinh	sinh.3	hyperbolic function	3
sqrt	sqrt.3	square root	1
tan	tan.3	trigonometric function	3
tanh	tanh.3	hyperbolic function	3
У0	j0.3	bessel function	???
у1	j0.3	bessel function	355
yn	j0.3	bessel function	355

NOTES

Properties of IEEE 754 Double-Precision

An **ulp** is one Unit in the Last Place.

The foregoing functions assume double–precision arithmetic conforming to the ${\tt IEEE}$ Standard 754 for Binary Floating–Point Arithmetic.

Wordsize:

64 bits, 8 bytes

Radix:

Binary

Precision:

53 significant bits, approximate to 16 significant decimals

If x and x' are consecutive positive Double–Precision numbers (they differ by 1 **ulp**), then $1.1e-16 < 0.5^{**}53 < (x'-x)/x \le 0.5^{**}52 < 2.3e-16$.

Range:

Overflow threshold = 2.0**102 = 1.8e308

Underflow threshold = $0.5^{**}1022 \approx 2.2e-308$

Overflow goes by default to a signed ∞

Underflow is Gradual, rounding to the nearest integer multiple of

$$0.5^{**}1074 \approx 4.9e-324.$$

Zero is represented ambiguously as +0 or -0:

Its sign transforms correctly through multiplication or division, and is preserved by addition of zeros with like signs; but x-x yields +0 for every finite x. The only operations that reveal zero's sign are division by zero and copysign(x,±0). In particular, comparison (x > y, $x \ge y$) cannot be affected by the sign of zero; but if finite x = y then $\infty = 1/(x-y) \ne -1/(y-x) = -\infty$

∞ is signed:

It persists when added to itself or to any finite number. Its sign transforms correctly through multiplication and division, (finite)/ $\pm \infty = \pm 0$ (nonzero)/0 = $\pm \infty$. But $\infty - \infty$, $\infty * 0$ and ∞ / ∞ are, like 0/0 and sqrt(-3), invalid operations that produce NaN.

Reserved operands:

There are $2^{**}53-2$ of them, all called NaN (Not a Number). Some, called Signaling NaNs, trap any floating–point operation performed upon them; they are used to mark missing or uninitialized values, or nonexistent elements of arrays. The rest are Quiet NaNs; they are the default results of Invalid Operations, and propagate through subsequent arithmetic operations. If $x \neq x$ then x is NaN; every other predicate (x > y, x = y, x < y, ...) is FALSE if NaN is involved.

56

ChorusOS 4.0

Mathematical Library math(3M)

NOTE: Trichotomy is violated by **NaN**. Besides being FALSE, predicates that entail ordered comparison, rather than mere (in)equality, signal Invalid Operation when **NaN** is involved.

Rounding:

Every algebraic operation (+, -, *, /, sqrt) is rounded by default to within half a **ulp**, and when the rounding error is exactly half a **ulp**, the rounded value's least significant bit is zero. This kind of rounding is usually the best kind, sometimes provably so; for instance, for every x = 1.0, 2.0, 3.0, 4.0, ..., 2.0**52, both (x/3.0)*3.0 == x and (x/10.0)*10.0 == x and ... despite both the quotients and the products having been rounded. No single kind of rounding can be proved best for every circumstance, IEEE 754 therefore provides rounding towards zero, or towards $+\infty$, or towards $-\infty$, at the programmer's option. The same kinds of rounding are specified for Binary–Decimal Conversions, at least for magnitudes between roughly 1.0e-10 and 1.0e37.

Exceptions:

IEEE 754 recognizes five kinds of floating-point exceptions, listed below in declining order of importance.

Exception Default Result Invalid Operation NaN, or FALSE $\pm \infty$ Divide by Zero $\pm \infty$ Underflow Gradual Underflow Inexact Rounded value

NOTE: An Exception is not an Error unless badly handled. What makes a class of exceptions exceptional is that no single default response can be satisfactory in every instance. On the other hand, if a default response will serve most instances satisfactorily, the unsatisfactory instances cannot justify aborting computation every time the exception occurs.

For each kind of floating–point exception, IEEE 754 provides a Flag that is raised each time its exception is signaled, and stays raised until the program resets it. Programs may also test, save and restore a flag. Thus, IEEE 754 provides three ways by which programs may cope with exceptions for which the default result might be unsatisfactory:

- 1. Test for a condition that might cause an exception later, and branch to avoid the exception.
- 2. Test a flag to see whether an exception has occurred since the program last reset its flag.

3. Test a result to see whether it is a value that only an exception could have produced.

CAUTION: The only reliable ways to discover whether Underflow has occurred are to test whether products or quotients lie closer to zero than the underflow threshold, or to test the Underflow flag. (Sums and differences cannot underflow in IEEE 754; if $x \neq y$ then x–y is correct to full precision and certainly nonzero regardless of how tiny it may be.) Products and quotients that underflow gradually can lose accuracy gradually without vanishing, comparing them with zero will not reveal the loss. If a gradually underflowed value is destined to be added to something bigger than the underflow threshold (as is almost always the case) digits lost to gradual underflow will not be missed because they would have been rounded off anyway. So gradual underflows are usually therefore *provably* ignorable. The same cannot be said of underflows flushed to 0.

At the option of an implementor conforming to IEEE 754, other ways to cope with exceptions may be provided:

- 4. ABORT. This mechanism classifies an exception in advance as an incident to be handled by means traditionally associated with error-handling statements like "ON ERROR GO TO ...". Different languages offer different forms of this statement, but most share the following characteristics:
- No means is provided to substitute a value for the offending operation's result and resume computation from what may be the middle of an expression. An exceptional result is abandoned.
- In a subprogram that lacks an error-handling statement, an exception causes
 the subprogram to abort within whatever program called it, and so on back
 up the chain of calling subprograms until an error-handling statement is
 encountered or the whole task is aborted and memory is dumped.
- 5. STOP. This mechanism, requiring an interactive debugging environment, is more for the programmer than the program. It classifies an exception in advance as a symptom of a programmer's error; the exception suspends execution as near as it can to the offending operation, so that the programmer can check to see how it happened. Quite often the first several exceptions turn out to be quite unexceptionable, so the programmer ought ideally to be able to resume execution after each one as if execution had not been stopped.
- 6. Other ways lie beyond the scope of this document.

RESTRICTIONS

This library implements very little of the above-mentionned IEEE 754 signaling requirements. As it does not rely upon the floating-point unit/operating system cooperation to signal errors asynchronously, it catches most errors explicitly. However, some NaN or results may be issued by the floating-point unit and be returned as such to the application without any warning better than the value of

Mathematical Library math(3M)

the result. Detected errors are reported by setting $\it errno$ to either ERANGE or EDOM, performing a system-dependant notification, and returning either + , - or NaN, whichever best suits the nature of the error. The above system-dependent notification is non-operational in ChorusOS products for which this library is currently distributed.

ATTRIBUTES

See attributes(5) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Evolving

pow(3M) Mathematical Library

NAME

exp, expm1, log, log10, log1p, pow - exponential, logarithm, power functions

SYNOPSIS

#include <math.h>
double exp(double x);

double expm1(double x);

double log(double x);

double log10(double x);

double log1p(double x);

double pow(double *x*, double *y*);

DESCRIPTION

The exp function computes the exponential value of the given argument x.

The *expm1* function computes the value $\exp(x)-1$ accurately even for extremely small values of x.

The *log* function computes the natural logarithm of the argument x.

The *log10* function computes the logarithm of argument *x* to base 10.

The log1p function computes the value of log(1+x) accurately even for extremely small values of x.

The *pow* computes the value of to the exponent

ERRORS (due to Roundoff)

The exp(x), log(x), expm1(x) and log1p(x) functions are accurate to within a ulp, and log10(x) to within approximately 2 ulp s. A ulp is one Unit in the Last Place. The error in pow(x, y) is below about 2 ulp s when its magnitude is moderate, but increases as pow(x, y) approaches the over/underflow thresholds. Almost as many bits as are occupied by the floating-point format's exponent field could be lost; that is 11 bits for IEEE 754 Double. The worst errors observed during testing have been under 300 ulp s for IEEE 754 Double. Moderate values of pow are sufficiently accurate that pow(integer, integer) is precise until it is greater than $2^{**}53$ for IEEE 754.

RETURN VALUES

These functions will return the appropriate computation unless an error occurs or an argument is out of range. The functions exp, expm1 and pow detect if the computed value will overflow, set errno to ERANGE and a system-dependent notification is performed. The pow(x, y) function checks whether x < 0 and y is not an integer; if this is true, errno is set to EDOM and a system-dependent notification is performed. errno is set to EDOM and a system-dependent notification is performed by log, unless x > 0, by log1p unless x > -1.

NOTES

The function pow(x, 0) returns $x^{**}0 = 1$ for all x including x = 0, , and NaN.

ATTRIBUTES

See attributes(5) for descriptions of the following attributes:

60

ChorusOS 4.0

Mathematical Library pow(3M)

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Evolving

SEE ALSO

 ${\tt math(3M)}$, ${\tt infnan(3M)}$

rint - round to closest integer functions

SYNOPSIS

#include <math.h>
double rint(double x);

DESCRIPTION

The *rint* function finds the integer (represented as a double precision number) nearest to *x* in the direction of the prevailing rounding mode.

NOTES

In the default rounding mode on a machine that conforms to IEEE 754, rint(x) is the integer closest to x, with the additional stipulation that if |rint(x)-x|=1/2, then rint(x) will be even. Other rounding modes can make rint perform like floor, or like ceil, or round up to zero.

Another way to obtain an integer near *x* is to declare (in C): double *x*; int k;k=*x*;

Most C compilers round x up to 0 to get the integer k, but not all. Use *floor*, *ceil*, or *rint* first, depending on the result required. Note that if x is larger than k can accommodate, the value of k and the presence or absence of an integer overflow are unpredictable.

ATTRIBUTES

See attributes(5) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Evolving

SEE ALSO

ceil(3M), floor(3M), ieee(3M), math(3M) abs(3STDC) fabs(3STDC)

62 ChorusOS 4.0 Last modified December 1999

Mathematical Library scalb(3M)

NAME

ieee, copysign, drem, finite, logb, scalb - IEEE 754 floating point support

SYNOPSIS

#include <math.h>

double copysign(double x, double y);

double drem(double x, double y);

int finite(double x);

double logb(double x);

double scalb(double x, int n);

DESCRIPTION

These functions are required for the ${\tt IEEE}$ 754 standard for floating-point arithmetic.

The *copysign* () function returns x with its sign changed to that of y.

The *drem ()* function returns the remainder $r := x - n^*y$ where n is the integer closest to the exact value of x/y. If |n - x/y| = 1/2, n is even. Consequently, the remainder is computed exactly and |r| <= |y| / 2. Note that *drem* (x, 0) is exceptional.

The *finite ()* function returns true (1) if the argument *x* is neither inifinity nor *NaN* value. Otherwise it returns false (0).

The *logb* () function computes the exponent of x, which is the integral part of log2 | x |, as a signed floating point value for non-zero x.

The *scalb* () function returns $x * (2^{**}n)$ computed, for integer n, without first computing $2^{**}n$.

RETURN VALUES

The IEEE 754 standard defines drem(x, 0) and drem(infinity, y) to be invalid operations that produce a **NaN**.

IEEE 754 defines logb (\pm infinity) = + infinity, and logb (0) = -HUGE_VAL, and requires the latter to signal Division–by–Zero. Upon successful completion, logb() returns the exponent of x.

ATTRIBUTES

See attributes(5) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Evolving

SEE ALSO

floor(3M), math(3M), infnan(3M)

NAME | sin – sine function

SYNOPSIS #include <math.h>

double sin(double x);

DESCRIPTION The *sin* function computes the sine of *x* (measured in radians). A large magnitude

argument may yield a result with little or no significance.

RETURN VALUES The *sin* function returns the sine value.

ATTRIBUTES See attributes(5) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Evolving

 $\textbf{SEE ALSO} \qquad \text{acos(3M), asin(3M), atan(3M), atan(3M), cos(3M), cosh(3M), sinh(3M),} \\$

tan(3M), tanh(3M), math(3M)

STANDARDS The *sin* function conforms to ANSI-C.

64 ChorusOS 4.0 Last modified December 1999

Mathematical Library sinh(3M)

NAME | sinh – hyperbolic sine function

SYNOPSIS #include <math.h>

double sinh(double x);

DESCRIPTION The *sinh* function computes the hyperbolic sine of x.

RETURN VALUES The *sinh* function returns the hyperbolic sine value unless the magnitude of *x*

is too large. In this event, errno is set to ERANGE and a system-dependent

notification is performed.

ATTRIBUTES See attributes(5) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Evolving

SEE ALSO acos(3M), asin(3M), atan(3M), atan(3M), atan(3M), cos(3M), cosh(3M), sin(3M),

tan(3M), tanh(3M), math(3M)

STANDARDS The *sinh* function conforms to ANSI-C.

NAME | sqrt, cbrt – cube root and square root functions

SYNOPSIS #include <math.h>

double cbrt(double x);

double sqrt(double x);

DESCRIPTION The *cbrt* function computes the cube root of x.

The *sqrt* function computes the non-negative square root of x.

RETURN VALUES The *cbrt* function returns the requested cube root. The *sqrt* function returns the

requested square root unless an error occurs. An attempt to take the *sqrt* of a negative value of *x* causes an error; in this event, *errno* is set to EDOM and a

system-dependent notification is performed.

ERRORS (due to Roundoff) The cbrt is

The cbrt function is accurate to within 0.7 ulps .

On a machine that conforms to IEEE 754 sqrt is correctly rounded in accordance with the rounding mode in force; the error is less than half a ulp in the default

mode (round-to-nearest). A ulp is one Unit in the Last Place carried.

ATTRIBUTES See attributes(5) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Evolving

Last modified December 1999

SEE ALSO

math(3M), infnan(3M)

STANDARDS

The *sqrt* function conforms to ANSI-C.

66 ChorusOS 4.0

Mathematical Library tan(3M)

NAME | tan – tangent function

SYNOPSIS #include <math.h>

double tan(double x);

DESCRIPTION The *tan* function computes the tangent of *x* (measured in radians). An argument

of high magnitude may yield a result with little or no significance. For a

discussion of errors due to rounding up, see math(3).

RETURN VALUES The *tan* function returns the tangent value.

ATTRIBUTES See attributes(5) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Evolving

SEE ALSO acos(3M), asin(3M), atan(3M), atan(3M), cos(3M), cosh(3M), sin(3M),

sinh(3M), tanh(3M), math(3M)

STANDARDS The *tan* function conforms to ANSI-C.

NAME | tanh – hyperbolic tangent function

SYNOPSIS #include <math.h> double tanh(double *x*);

DESCRIPTION The *tanh* function computes the hyperbolic tangent of *x*. For a discussion of

errors due to rounding up, see math(3).

RETURN VALUES The *tanh* function returns the hyperbolic tangent value.

ATTRIBUTES See attributes(5) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Evolving

SEE ALSO acos(3M), asin(3M), atan(3M), atan2(3M), cos(3M), cosh(3M), sin(3M),

sinh(3M), tan(3M), math(3M)

STANDARDS The *tanh* function conforms to ANSI-C.

68 ChorusOS 4.0 Last modified December 1999

Mathematical Library y0(3M)

NAME

j0, j1, jn, y0, y1, yn – Bessel functions of the first and second kind

SYNOPSIS

#include <math.h>
double jo(double x);

double j1(double x);

double jn(int n, double x);

double y0(double x);

double **y1**(double *x*);

double yn(int n, double x);

DESCRIPTION

The j0 and j1 functions compute the Bessel function of the first type of the order 0 and the order 1, respectively, for the real value x. The jn function computes the Bessel function of the first type of the integer order n for the real value x.

RETURN VALUES

If these functions are successful, the computed value is returned. Upon successful completion, j00, j10 and jn0 return the relevant Bessel value of x of the first type.

Upon successful completion, y00, y10 and yn0 return the relevant Bessel value of x of the second type. If the x argument of y00, y10 or yn0 is negative, -huge_val is returned and errno is set errno is set to EDOM. If x is 0.0, -huge_val is returned and errno is set to ERANGE. If the correct result would cause overflow, -huge_val or huge_val is returned and errno is set to ERANGE

ATTRIBUTES

See attributes(5) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Evolving

SEE ALSO

math(3M), infnan(3M)

j0, j1, jn, y0, y1, yn - Bessel functions of the first and second kind

SYNOPSIS

#include <math.h>

double j0(double x);
double j1(double x);

double jn(int n, double x);

double y0(double x);

double **y1**(double *x*);

double yn(int n, double x);

DESCRIPTION

The j0 and j1 functions compute the Bessel function of the first type of the order 0 and the order 1, respectively, for the real value x. The jn function computes the Bessel function of the first type of the integer order n for the real value x.

RETURN VALUES

If these functions are successful, the computed value is returned. Upon successful completion, j0(), j1() and jn() return the relevant Bessel value of x of the first type.

Upon successful completion, y00, y10 and yn0 return the relevant Bessel value of x of the second type. If the x argument of y00, y10 or yn0 is negative, -huge_val is returned and errno is set to EDOM. If x is 0.0, -huge_val is returned and errno is set to ERANGE. If the correct result would cause overflow, -huge_val or huge_val is returned and errno is set to ERANGE

ATTRIBUTES

See attributes(5) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Evolving

SEE ALSO

math(3M), infnan(3M)

70

ChorusOS 4.0

Mathematical Library yn(3M)

NAME

j0, j1, jn, y0, y1, yn – Bessel functions of the first and second kind

SYNOPSIS

#include <math.h>
double jo(double x);

double j1(double x);

double jn(int n, double x);

double y0(double x);

double **y1**(double *x*);

double **yn**(int *n*, double *x*);

DESCRIPTION

The j0 and j1 functions compute the Bessel function of the first type of the order 0 and the order 1, respectively, for the real value x. The jn function computes the Bessel function of the first type of the integer order n for the real value x.

RETURN VALUES

If these functions are successful, the computed value is returned. Upon successful completion, j00, j10 and jn0 return the relevant Bessel value of x of the first type.

Upon successful completion, y00, y10 and yn0 return the relevant Bessel value of x of the second type. If the x argument of y00, y10 or yn0 is negative, -huge_val is returned and errno is set errno is set to EDOM. If x is 0.0, -huge_val is returned and errno is set to ERANGE. If the correct result would cause overflow, -huge_val or huge_val is returned and errno is set to ERANGE

ATTRIBUTES

See attributes(5) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Evolving

SEE ALSO

math(3M), infnan(3M)

72 ChorusOS 4.0

Index

A	erfc — error function operators 29–30
acos — arc cosine function 15	exp — exponential, logarithm, power
acosh — inverse hyperbolic cosine function 16	functions 31, 33, 48, 50, 52, 60
asin — arc sine function 17	expm1 — exponential, logarithm, power
asinh — inverse hyperbolic sine function 18	functions 31, 33, 48, 50, 52, 60
atan — arc tangent function of one variable 20	
atan2 — arc tangent function of two	F
variables 19	finite — IEEE 754 floating point support 25,
atanh — inverse hyperbolic tangent	28, 35, 41, 54, 63
function 21	floor — largest integral value not greater than x 36
C	fmod — floating-point remainder function 37
cabs — euclidean distance and complex	
absolute value functions 22,	G
40	gamma — log gamma function, gamma
cbrt — cube root and square root functions 23, 66	function 38, 46
ceil — smallest integral value not less than x 24	Н
copysign — IEEE 754 floating point support 25, 28, 35, 41, 54, 63	hypot — euclidean distance and complex absolute value functions 22,
cos — cosine function 26	40
cosh — hyperbolic cosine function 27	10
•	I
D	_
drem — IEEE 754 floating point support 25, 28, 35, 41, 54, 63	infnan — signals invalid floating point operations 42
	J
E	j0 — Bessel functions of the first and second
erf — error function operators 29–30	types 43–45, 69–71
	Inday 72

Index-73

j1 — Bessel functions of the first and second types 43–45, 69–71 jn — Bessel functions of the first and second types 43–45, 69–71

L

lgamma — log gamma function, gamma function 38, 46
log — exponential, logarithm, power functions 31, 33, 48, 50, 52, 60
log10 — exponential, logarithm, power functions 31, 33, 48, 50, 52, 60
log1p — exponential, logarithm, power functions 31, 33, 48, 50, 52, 60
logb — IEEE 754 floating point support 25, 28, 35, 41, 54, 63

M

math — introduction to mathematical library functions 55

P

pow — exponential, logarithm, power functions 31, 33, 48, 50, 52, 60

R

rint — round to closest integer functions 62

S

scalb — IEEE 754 floating point support 25, 28, 35, 41, 54, 63 sin — sine function 64 sinh — hyperbolic sine function 65 sqrt — cube root and square root functions 23, 66

T

tan — tangent function 67 tanh — hyperbolic tangent function 68

Y

 $\begin{array}{c} y0 - \text{Bessel functions of the first and second} \\ types \quad 43\text{-}45, \, 69\text{-}71 \\ y1 - \text{Bessel functions of the first and second} \\ types \quad 43\text{-}45, \, 69\text{-}71 \\ yn - \text{Bessel functions of the first and second} \\ types \quad 43\text{-}45, \, 69\text{-}71 \\ \end{array}$