```
**********************************************************
   18837 Tue Jun  4 01:04:25 2019
new/usr/src/head/iso/math_c99.h
11175 libm should use signbit() correctly
11188 c99 math macros should return strictly backward compatible values
**********************************************************
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License (the "License").
 * You may not use this file except in compliance with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" replaced with your own identifying
 * information: Portions Copyright [yyyy] [name of copyright owner]
 *
 * CDDL HEADER END
 */
/*
 * Copyright 2011 Nexenta Systems, Inc.  All rights reserved.
 */
/*
 * Copyright 2005 Sun Microsystems, Inc.  All rights reserved.
 * Use is subject to license terms.
 */

#ifndef _ISO_MATH_C99_H
#define _ISO_MATH_C99_H

#include <sys/isa_defs.h>
#include <sys/feature_tests.h>

#ifdef __cplusplus
extern "C" {
#endif

#undef  FP_ZERO
#define FP_ZERO         0
#undef  FP_SUBNORMAL
#define FP_SUBNORMAL    1
#undef  FP_NORMAL
#define FP_NORMAL       2
#undef  FP_INFINITE
#define FP_INFINITE     3
#undef  FP_NAN
#define FP_NAN          4

#if defined(_STDC_C99) || _XOPEN_SOURCE - 0 >= 600 || defined(__C99FEATURES__)
#if defined(__GNUC__)
#undef  HUGE_VAL
#define HUGE_VAL        (__builtin_huge_val())
#undef  HUGE_VALF
#define HUGE_VALF       (__builtin_huge_valf())
#undef  HUGE_VALL
#define HUGE_VALL       (__builtin_huge_vall())
#undef  INFINITY
#define INFINITY        (__builtin_inff())
#undef  NAN
```

```
#define NAN             (__builtin_nanf(""))

/*
 * C99 7.12.3 classification macros
 */
#undef  isnan
#undef  isinf
#if __GNUC__ >= 4
#define fpclassify(x)   __builtin_fpclassify(FP_NAN, FP_INFINITE, FP_NORMAL, \
    FP_SUBNORMAL, FP_ZERO, x)
#endif /* ! codereview */
#define isnan(x)        __builtin_isnan(x)
#define isinf(x)        __builtin_isinf(x)
#define isfinite(x)     (__builtin_isfinite(x) != 0)
#define isnormal(x)     (__builtin_isnormal(x) != 0)
#define signbit(x)      (__builtin_signbit(x) != 0)
#define fpclassify(x)   __builtin_fpclassify(FP_NAN, FP_INFINITE, FP_NORMAL, \
    FP_SUBNORMAL, FP_ZERO, x)
#define isfinite(x)     __builtin_isfinite(x)
#define isnormal(x)     __builtin_isnormal(x)
#define signbit(x)      (__builtin_signbit(x) > 0)
#else  /* __GNUC__ >= 4 */
#define isnan(x)        __extension__( \
                        { __typeof(x) __x_n = (x); \
                        __builtin_isunordered(__x_n, __x_n); })
#define isinf(x)        __extension__( \
                        { __typeof(x) __x_i = (x); \
                        __x_i == (__typeof(__x_i)) INFINITY || \
                        __x_i == (__typeof(__x_i)) (-INFINITY); })
#undef  isfinite
#define isfinite(x)     __extension__( \
                        { __typeof(x) __x_f = (x); \
                        !isnan(__x_f) && !isinf(__x_f); })
#undef  isnormal
#define isnormal(x)     __extension__( \
                        { __typeof(x) __x_r = (x); isfinite(__x_r) && \
                        (sizeof (__x_r) == sizeof (float) ? \
                        __builtin_fabsf(__x_r) >= __FLT_MIN__ : \
                        sizeof (__x_r) == sizeof (double) ? \
                        __builtin_fabs(__x_r) >= __DBL_MIN__ : \
                        __builtin_fabsl(__x_r) >= __LDBL_MIN__); })
#undef  fpclassify
#define fpclassify(x)   __extension__( \
                        { __typeof(x) __x_c = (x); \
                        isnan(__x_c) ? FP_NAN : \
                        isinf(__x_c) ? FP_INFINITE : \
                        isnormal(__x_c) ? FP_NORMAL : \
                        __x_c == (__typeof(__x_c)) 0 ? FP_ZERO : \
                        FP_SUBNORMAL; })
#undef  signbit
#if defined(_BIG_ENDIAN)
#define signbit(x)      __extension__( \
                        { __typeof(x) __x_s = (x); \
                        (int)(*(unsigned *)&__x_s >> 31); })
#elif defined(_LITTLE_ENDIAN)
#define signbit(x)      __extension__( \
                        { __typeof(x) __x_s = (x); \
                        (sizeof (__x_s) == sizeof (float) ? \
                        (int)(*(unsigned *)&__x_s >> 31) : \
                        sizeof (__x_s) == sizeof (double) ? \
                        (int)(((unsigned *)&__x_s)[1] >> 31) : \
                        (int)(((unsigned short *)&__x_s)[4] >> 15)); })
#endif  /* defined(_BIG_ENDIAN) */
#endif  /* __GNUC__ >= 4 */

/*
```

```
 * C99 7.12.14 comparison macros
 */
#undef  isgreater
#define isgreater(x, y)         __builtin_isgreater(x, y)
#undef  isgreaterequal
#define isgreaterequal(x, y)    __builtin_isgreaterequal(x, y)
#undef  isless
#define isless(x, y)            __builtin_isless(x, y)
#undef  islessequal
#define islessequal(x, y)       __builtin_islessequal(x, y)
#undef  islessgreater
#define islessgreater(x, y)     __builtin_islessgreater(x, y)
#undef  isunordered
#define isunordered(x, y)       __builtin_isunordered(x, y)
#else   /* defined(__GNUC__) */
#undef  HUGE_VAL
#define HUGE_VAL        __builtin_huge_val
#undef  HUGE_VALF
#define HUGE_VALF       __builtin_huge_valf
#undef  HUGE_VALL
#define HUGE_VALL       __builtin_huge_vall
#undef  INFINITY
#define INFINITY        __builtin_infinity
#undef  NAN
#define NAN             __builtin_nan

/*
 * C99 7.12.3 classification macros
 */
#undef  fpclassify
#define fpclassify(x)   __builtin_fpclassify(x)
#undef  isfinite
#define isfinite(x)     __builtin_isfinite(x)
#undef  isinf
#define isinf(x)        __builtin_isinf(x)
#undef  isnan
#define isnan(x)        __builtin_isnan(x)
#undef  isnormal
#define isnormal(x)     __builtin_isnormal(x)
#undef  signbit
#define signbit(x)      __builtin_signbit(x)

/*
 * C99 7.12.14 comparison macros
 */
#undef  isgreater
#define isgreater(x, y)         ((x) __builtin_isgreater(y))
#undef  isgreaterequal
#define isgreaterequal(x, y)    ((x) __builtin_isgreaterequal(y))
#undef  isless
#define isless(x, y)            ((x) __builtin_isless(y))
#undef  islessequal
#define islessequal(x, y)       ((x) __builtin_islessequal(y))
#undef  islessgreater
#define islessgreater(x, y)     ((x) __builtin_islessgreater(y))
#undef  isunordered
#define isunordered(x, y)       ((x) __builtin_isunordered(y))
#endif  /* defined(__GNUC__) */
#endif  /* defined(_STDC_C99) || _XOPEN_SOURCE - 0 >= 600 || ... */

#if defined(__EXTENSIONS__) || defined(_STDC_C99) || \
        (!defined(_STRICT_STDC) && !defined(__XOPEN_OR_POSIX)) || \
        defined(__C99FEATURES__)
#if defined(__FLT_EVAL_METHOD__) && __FLT_EVAL_METHOD__ - 0 == 0
typedef float float_t;
typedef double double_t;
```

```
#elif __FLT_EVAL_METHOD__ - 0 == 1
typedef double float_t;
typedef double double_t;
#elif __FLT_EVAL_METHOD__ - 0 == 2
typedef long double float_t;
typedef long double double_t;
#elif defined(__sparc) || defined(__amd64)
typedef float float_t;
typedef double double_t;
#elif defined(__i386)
typedef long double float_t;
typedef long double double_t;
#endif

#undef  FP_ILOGB0
#define FP_ILOGB0       (-2147483647)
#undef  FP_ILOGBNAN
#define FP_ILOGBNAN     2147483647

#undef  MATH_ERRNO
#define MATH_ERRNO      1
#undef  MATH_ERREXCEPT
#define MATH_ERREXCEPT  2
#undef  math_errhandling
#define math_errhandling        MATH_ERREXCEPT

extern double acosh(double);
extern double asinh(double);
extern double atanh(double);

extern double exp2(double);
extern double expm1(double);
extern int ilogb(double);
extern double log1p(double);
extern double log2(double);
extern double logb(double);
extern double scalbn(double, int);
extern double scalbln(double, long int);

extern double cbrt(double);
extern double hypot(double, double);

extern double erf(double);
extern double erfc(double);
extern double lgamma(double);
extern double tgamma(double);

extern double nearbyint(double);
extern double rint(double);
extern long int lrint(double);
extern double round(double);
extern long int lround(double);
extern double trunc(double);

extern double remainder(double, double);
extern double remquo(double, double, int *);

extern double copysign(double, double);
extern double nan(const char *);
extern double nextafter(double, double);
extern double nexttoward(double, long double);

extern double fdim(double, double);
extern double fmax(double, double);
extern double fmin(double, double);
extern double fma(double, double, double);

extern float acosf(float);
extern float asinf(float);
extern float atanf(float);
extern float atan2f(float, float);
extern float cosf(float);
extern float sinf(float);
extern float tanf(float);

extern float acoshf(float);
extern float asinhf(float);
extern float atanhf(float);
extern float coshf(float);
extern float sinhf(float);
extern float tanhf(float);

extern float expf(float);
extern float exp2f(float);
extern float expm1f(float);
extern float frexpf(float, int *);
extern int ilogbf(float);
extern float ldexpf(float, int);
extern float logf(float);
extern float log10f(float);
extern float log1pf(float);
extern float log2f(float);
extern float logbf(float);
extern float modff(float, float *);
extern float scalbnf(float, int);
extern float scalblnf(float, long int);

extern float cbrtf(float);
extern float fabsf(float);
extern float hypotf(float, float);
extern float powf(float, float);
extern float sqrtf(float);

extern float erff(float);
extern float erfcf(float);
extern float lgammaf(float);
extern float tgammaf(float);

extern float ceilf(float);
extern float floorf(float);
extern float nearbyintf(float);
extern float rintf(float);
extern long int lrintf(float);
extern float roundf(float);
extern long int lroundf(float);
extern float truncf(float);

extern float fmodf(float, float);
extern float remainderf(float, float);
extern float remquof(float, float, int *);

extern float copysignf(float, float);
extern float nanf(const char *);
extern float nextafterf(float, float);
extern float nexttowardf(float, long double);

extern float fdimf(float, float);
extern float fmaxf(float, float);
extern float fminf(float, float);

extern float fmaf(float, float, float);
```

```
extern long double acosl(long double);
extern long double asinl(long double);
extern long double atanl(long double);
extern long double atan2l(long double, long double);
extern long double cosl(long double);
extern long double sinl(long double);
extern long double tanl(long double);

extern long double acoshl(long double);
extern long double asinhl(long double);
extern long double atanhl(long double);
extern long double coshl(long double);
extern long double sinhl(long double);
extern long double tanhl(long double);

extern long double expl(long double);
extern long double exp2l(long double);
extern long double expm1l(long double);
extern long double frexpl(long double, int *);
extern int ilogbl(long double);
extern long double ldexpl(long double, int);
extern long double logl(long double);
extern long double log10l(long double);
extern long double log1pl(long double);
extern long double log2l(long double);
extern long double logbl(long double);
extern long double modfl(long double, long double *);
extern long double scalbnl(long double, int);
extern long double scalblnl(long double, long int);

extern long double cbrtl(long double);
extern long double fabsl(long double);
extern long double hypotl(long double, long double);
extern long double powl(long double, long double);
extern long double sqrtl(long double);

extern long double erfl(long double);
extern long double erfcl(long double);
extern long double lgammal(long double);
extern long double tgammal(long double);

extern long double ceill(long double);
extern long double floorl(long double);
extern long double nearbyintl(long double);
extern long double rintl(long double);
extern long int lrintl(long double);
extern long double roundl(long double);
extern long int lroundl(long double);
extern long double truncl(long double);

extern long double fmodl(long double, long double);
extern long double remainderl(long double, long double);
extern long double remquol(long double, long double, int *);

extern long double copysignl(long double, long double);
extern long double nanl(const char *);
extern long double nextafterl(long double, long double);
extern long double nexttowardl(long double, long double);

extern long double fdiml(long double, long double);
extern long double fmaxl(long double, long double);
extern long double fminl(long double, long double);

extern long double fmal(long double, long double, long double);
```

```
#if !defined(_STRICT_STDC) && !defined(_NO_LONGLONG) || defined(_STDC_C99) || \
        defined(__C99FEATURES__)
extern long long int llrint(double);
extern long long int llround(double);

extern long long int llrintf(float);
extern long long int llroundf(float);

extern long long int llrintl(long double);
extern long long int llroundl(long double);
#endif

#if !defined(__cplusplus)
#pragma does_not_read_global_data(asinh, exp2, expm1)
#pragma does_not_read_global_data(ilogb, log2)
#pragma does_not_read_global_data(scalbn, scalbln, cbrt)
#pragma does_not_read_global_data(erf, erfc, tgamma)
#pragma does_not_read_global_data(nearbyint, rint, lrint, round, lround, trunc)
#pragma does_not_read_global_data(remquo)
#pragma does_not_read_global_data(copysign, nan, nexttoward)
#pragma does_not_read_global_data(fdim, fmax, fmin, fma)
#pragma does_not_write_global_data(asinh, exp2, expm1)
#pragma does_not_write_global_data(ilogb, log2)
#pragma does_not_write_global_data(scalbn, scalbln, cbrt)
#pragma does_not_write_global_data(erf, erfc, tgamma)
#pragma does_not_write_global_data(nearbyint, rint, lrint, round, lround, trunc)
#pragma does_not_write_global_data(copysign, nan, nexttoward)
#pragma does_not_write_global_data(fdim, fmax, fmin, fma)

#pragma does_not_read_global_data(acosf, asinf, atanf, atan2f)
#pragma does_not_read_global_data(cosf, sinf, tanf)
#pragma does_not_read_global_data(acoshf, asinhf, atanhf, coshf, sinhf, tanhf)
#pragma does_not_read_global_data(expf, exp2f, expm1f, frexpf, ilogbf, ldexpf)
#pragma does_not_read_global_data(logf, log10f, log1pf, log2f, logbf)
#pragma does_not_read_global_data(modff, scalbnf, scalblnf)
#pragma does_not_read_global_data(cbrtf, fabsf, hypotf, powf, sqrtf)
#pragma does_not_read_global_data(erff, erfcf, lgammaf, tgammaf)
#pragma does_not_read_global_data(ceilf, floorf, nearbyintf)
#pragma does_not_read_global_data(rintf, lrintf, roundf, lroundf, truncf)
#pragma does_not_read_global_data(fmodf, remainderf, remquof)
#pragma does_not_read_global_data(copysignf, nanf, nextafterf, nexttowardf)
#pragma does_not_read_global_data(fdimf, fmaxf, fminf, fmaf)
#pragma does_not_write_global_data(acosf, asinf, atanf, atan2f)
#pragma does_not_write_global_data(cosf, sinf, tanf)
#pragma does_not_write_global_data(acoshf, asinhf, atanhf, coshf, sinhf, tanhf)
#pragma does_not_write_global_data(expf, exp2f, expm1f, ilogbf, ldexpf)
#pragma does_not_write_global_data(logf, log10f, log1pf, log2f, logbf)
#pragma does_not_write_global_data(cbrtf, fabsf, hypotf, powf, sqrtf)
#pragma does_not_write_global_data(erff, erfcf, tgammaf)
#pragma does_not_write_global_data(ceilf, floorf, nearbyintf)
#pragma does_not_write_global_data(rintf, lrintf, roundf, lroundf, truncf)
#pragma does_not_write_global_data(fmodf, remainderf)
#pragma does_not_write_global_data(copysignf, nanf, nextafterf, nexttowardf)
#pragma does_not_write_global_data(fdimf, fmaxf, fminf, fmaf)

#pragma does_not_read_global_data(acosl, asinl, atanl, atan2l)
#pragma does_not_read_global_data(cosl, sinl, tanl)
#pragma does_not_read_global_data(acoshl, asinhl, atanhl, coshl, sinhl, tanhl)
#pragma does_not_read_global_data(expl, exp2l, expm1l, frexpl, ilogbl, ldexpl)
#pragma does_not_read_global_data(logl, log10l, log1pl, log2l, logbl)
#pragma does_not_read_global_data(modfl, scalbnl, scalblnl)
#pragma does_not_read_global_data(cbrtl, fabsl, hypotl, powl, sqrtl)
#pragma does_not_read_global_data(erfl, erfcl, lgammal, tgammal)
#pragma does_not_read_global_data(ceill, floorl, nearbyintl)
#pragma does_not_read_global_data(rintl, lrintl, roundl, lroundl, truncl)
#pragma does_not_read_global_data(fmodl, remainderl, remquol)
```

```
#pragma does_not_read_global_data(copysignl, nanl, nextafterl, nexttowardl)
#pragma does_not_read_global_data(fdiml, fmaxl, fminl, fmal)
#pragma does_not_write_global_data(acosl, asinl, atanl, atan2l)
#pragma does_not_write_global_data(cosl, sinl, tanl)
#pragma does_not_write_global_data(acoshl, asinhl, atanhl, coshl, sinhl, tanhl)
#pragma does_not_write_global_data(expl, exp2l, expm1l, ilogbl, ldexpl)
#pragma does_not_write_global_data(logl, log10l, log1pl, log2l, logbl)
#pragma does_not_write_global_data(cbrtl, fabsl, hypotl, powl, sqrtl)
#pragma does_not_write_global_data(erfl, erfcl, tgammal)
#pragma does_not_write_global_data(ceill, floorl, nearbyintl)
#pragma does_not_write_global_data(rintl, lrintl, roundl, lroundl, truncl)
#pragma does_not_write_global_data(fmodl, remainderl)
#pragma does_not_write_global_data(copysignl, nanl, nextafterl, nexttowardl)
#pragma does_not_write_global_data(fdiml, fmaxl, fminl, fmal)

#if !defined(_STRICT_STDC) && !defined(_NO_LONGLONG) || defined(_STDC_C99) || \
        defined(__C99FEATURES__)
#pragma does_not_read_global_data(llrint, llround)
#pragma does_not_read_global_data(llrintf, llroundf, llrintl, llroundl)
#pragma does_not_write_global_data(llrint, llround)
#pragma does_not_write_global_data(llrintf, llroundf, llrintl, llroundl)
#endif
#endif  /* !defined(__cplusplus) */

#if defined(__MATHERR_ERRNO_DONTCARE)
#pragma does_not_read_global_data(acosh, atanh, hypot, lgamma, log1p, logb)
#pragma does_not_read_global_data(nextafter, remainder)
#pragma does_not_write_global_data(acosh, atanh, hypot, log1p, logb)
#pragma does_not_write_global_data(nextafter, remainder)

#pragma no_side_effect(acosh, asinh, atanh, exp2, expm1)
#pragma no_side_effect(ilogb, log1p, log2, logb)
#pragma no_side_effect(scalbn, scalbln, cbrt, hypot)
#pragma no_side_effect(erf, erfc, tgamma)
#pragma no_side_effect(nearbyint, rint, lrint, round, lround, trunc)
#pragma no_side_effect(remainder)
#pragma no_side_effect(copysign, nan, nextafter, nexttoward)
#pragma no_side_effect(fdim, fmax, fmin, fma)

#pragma no_side_effect(acosf, asinf, atanf, atan2f)
#pragma no_side_effect(cosf, sinf, tanf, coshf, sinhf, tanhf)
#pragma no_side_effect(acoshf, asinhf, atanhf, coshf, sinhf, tanhf)
#pragma no_side_effect(expf, exp2f, expm1f, ilogbf, ldexpf)
#pragma no_side_effect(logf, log10f, log1pf, log2f, logbf)
#pragma no_side_effect(cbrtf, fabsf, hypotf, powf, sqrtf)
#pragma no_side_effect(erff, erfcf, tgammaf)
#pragma no_side_effect(ceilf, floorf, nearbyintf)
#pragma no_side_effect(rintf, lrintf, roundf, lroundf, truncf)
#pragma no_side_effect(fmodf, remainderf)
#pragma no_side_effect(copysignf, nanf, nextafterf, nexttowardf)
#pragma no_side_effect(fdimf, fmaxf, fminf, fmaf)

#pragma no_side_effect(acosl, asinl, atanl, atan2l)
#pragma no_side_effect(cosl, sinl, tanl, coshl, sinhl, tanhl)
#pragma no_side_effect(acoshl, asinhl, atanhl, coshl, sinhl, tanhl)
#pragma no_side_effect(expl, exp2l, expm1l, ilogbl, ldexpl)
#pragma no_side_effect(logl, log10l, log1pl, log2l, logbl)
#pragma no_side_effect(cbrtl, fabsl, hypotl, powl, sqrtl)
#pragma no_side_effect(erfl, erfcl, tgammal)
#pragma no_side_effect(ceill, floorl, nearbyintl)
#pragma no_side_effect(rintl, lrintl, roundl, lroundl, truncl)
#pragma no_side_effect(fmodl, remainderl)
#pragma no_side_effect(copysignl, nanl, nextafterl, nexttowardl)
#pragma no_side_effect(fdiml, fmaxl, fminl, fmal)

#if !defined(_STRICT_STDC) && !defined(_NO_LONGLONG) || defined(_STDC_C99) || \
        defined(__C99FEATURES__)
#pragma no_side_effect(llrint, llround, llrintf, llroundf, llrintl, llroundl)
#endif
#endif  /* defined(__MATHERR_ERRNO_DONTCARE) */
#endif  /* defined(__EXTENSIONS__) || defined(_STDC_C99) || ... */

#ifdef __cplusplus
}
______unchanged_portion_omitted_
```

```
**********************************************************
    7495 Tue Jun  4 01:04:26 2019
new/usr/src/lib/libm/common/C/jn.c
11175 libm should use signbit() correctly
11188 c99 math macros should return strictly backward compatible values
**********************************************************
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License (the "License").
 * You may not use this file except in compliance with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" replaced with your own identifying
 * information: Portions Copyright [yyyy] [name of copyright owner]
 *
 * CDDL HEADER END
 */

/*
 * Copyright 2011 Nexenta Systems, Inc.  All rights reserved.
 */
/*
 * Copyright 2006 Sun Microsystems, Inc.  All rights reserved.
 * Use is subject to license terms.
 */

#pragma weak __jn = jn
#pragma weak __yn = yn

/*
 * floating point Bessel's function of the 1st and 2nd kind
 * of order n: jn(n,x),yn(n,x);
 *
 * Special cases:
 *      y0(0)=y1(0)=yn(n,0) = -inf with division by zero signal;
 *      y0(-ve)=y1(-ve)=yn(n,-ve) are NaN with invalid signal.
 * Note 2. About jn(n,x), yn(n,x)
 *      For n=0, j0(x) is called,
 *      for n=1, j1(x) is called,
 *      for n<x, forward recursion us used starting
 *      from values of j0(x) and j1(x).
 *      for n>x, a continued fraction approximation to
 *      j(n,x)/j(n-1,x) is evaluated and then backward
 *      recursion is used starting from a supposed value
 *      for j(n,x). The resulting value of j(0,x) is
 *      compared with the actual value to correct the
 *      supposed value of j(n,x).
 *
 *      yn(n,x) is similar in all respects, except
 *      that forward recursion is used for all
 *      values of n>1.
 *
 */

#include "libm.h"
#include <float.h>      /* DBL_MIN */
#include <values.h>     /* X_TLOSS */
```

```
#include "xpg6.h"       /* __xpg6 */

#define GENERIC double

static const GENERIC
        invsqrtpi = 5.641895835477562869480794515607725858441e-0001,
        two     = 2.0,
        zero    = 0.0,
        one     = 1.0;

GENERIC
jn(int n, GENERIC x)
{
jn(int n, GENERIC x) {
        int i, sgn;
        GENERIC a, b, temp = 0;
        GENERIC z, w, ox, on;

        /*
         * J(-n,x) = (-1)^n * J(n, x), J(n, -x) = (-1)^n * J(n, x)
         * Thus, J(-n,x) = J(n,-x)
         */
        ox = x;
        on = (GENERIC)n;

        ox = x; on = (GENERIC)n;
        if (n < 0) {
                n = -n;
                x = -x;
        }
        if (isnan(x))
                return (x*x);   /* + -> * for Cheetah */
        if (!((int)_lib_version == libm_ieee ||
        if (!((int) _lib_version == libm_ieee ||
            (__xpg6 & _C99SUSv3_math_errexcept) != 0)) {
                if (fabs(x) > X_TLOSS)
                        return (_SVID_libm_err(on, ox, 38));
        }
        if (n == 0)
                return (j0(x));
        if (n == 1)
                return (j1(x));
        if ((n&1) == 0)
                sgn = 0;                        /* even n */
        else
                sgn = signbit(x);       /* old n  */
        x = fabs(x);
        if (x == zero||!finite(x)) b = zero;
        else if ((GENERIC)n <= x) {
                                        /*
                                         * Safe to use
                                         *  J(n+1,x)=2n/x *J(n,x)-J(n-1,x)
                                         */
                if (x > 1.0e91) {
                                /*
                                 * x >> n**2
                                 *    Jn(x) = cos(x-(2n+1)*pi/4)*sqrt(2/x*pi)
                                 *   Yn(x) = sin(x-(2n+1)*pi/4)*sqrt(2/x*pi)
                                 *   Let s=sin(x), c=cos(x),
                                 *      xn=x-(2n+1)*pi/4, sqt2 = sqrt(2),then
                                 *
                                 *         n    sin(xn)*sqt2    cos(xn)*sqt2
                                 *      ----------------------------------
                                 *         0     s-c             c+s
                                 *         1    -s-c            -c+s
                                 *         2    -s+c            -c-s
                                 *         3     s+c             c-s
                                 */
                        switch (n&3) {
                        case 0:
                                temp =  cos(x)+sin(x);
                                break;
                        case 1:
                                temp = -cos(x)+sin(x);
                                break;
                        case 2:
                                temp = -cos(x)-sin(x);
                                break;
                        case 3:
                                temp =  cos(x)-sin(x);
                                break;
                    case 0: temp =  cos(x)+sin(x); break;
                    case 1: temp = -cos(x)+sin(x); break;
                    case 2: temp = -cos(x)-sin(x); break;
                    case 3: temp =  cos(x)-sin(x); break;
                        }
                        b = invsqrtpi*temp/sqrt(x);
                } else {
                        a = j0(x);
                        b = j1(x);
                        for (i = 1; i < n; i++) {
                                temp = b;
                                /* avoid underflow */
                                b = b*((GENERIC)(i+i)/x) - a;
                    b = b*((GENERIC)(i+i)/x) - a; /* avoid underflow */
                                a = temp;
                        }
                }
        } else {
                if (x < 1e-9) { /* use J(n,x) = 1/n!*(x/2)^n */
                        b = pow(0.5*x, (GENERIC) n);
                        if (b != zero) {
                                for (a = one, i = 1; i <= n; i++)
                                        a *= (GENERIC)i;
                    for (a = one, i = 1; i <= n; i++) a *= (GENERIC)i;
                                b = b/a;
                        }
                } else {
                        /*
                         * use backward recurrence
                         *                      x         x^2     x^2
                         *  J(n,x)/J(n-1,x) =  ----   ------   ------   .....
                         *                      2n  - 2(n+1) - 2(n+2)
                         *
                         *                      1         1         1
                         *  (for large x)   =  ----  ------   ------   .....
                         *                      2n   2(n+1)   2(n+2)
                         *                      -- - ------ - ------ -
                         *                       x       x             x
                         *
                         * Let w = 2n/x and h = 2/x, then the above quotient
                         * is equal to the continued fraction:
                         *                  1
                         *      = -----------------------
                         *                         1
                         *         w - -----------------
                         *                        1
                         *                      w+h - ---------
                         *                         w+2h - ...
                         *
                         * To determine how many terms needed, let
                         * Q(0) = w, Q(1) = w(w+h) - 1,
```

```
                         * Q(k) = (w+k*h)*Q(k-1) - Q(k-2),
                         * When Q(k) > 1e4      good for single
                         * When Q(k) > 1e9      good for double
                         * When Q(k) > 1e17     good for quaduple
                         */
                        /* determine k */
            /* determin k */
                        GENERIC t, v;
                        double q0, q1, h, tmp;
                        int k, m;
                        w  = (n+n)/(double)x;
                        h = 2.0/(double)x;
                        q0 = w;
                        z = w + h;
                        q1 = w*z - 1.0;
                        k = 1;

                double q0, q1, h, tmp; int k, m;
                w  = (n+n)/(double)x; h = 2.0/(double)x;
                q0 = w;  z = w + h; q1 = w*z - 1.0; k = 1;
                        while (q1 < 1.0e9) {
                                k += 1;
                                z += h;
                        k += 1; z += h;
                                tmp = z*q1 - q0;
                                q0 = q1;
                                q1 = tmp;
                        }
                        m = n+n;
                        for (t = zero, i = 2*(n+k); i >= m; i -= 2)
                                t = one/(i/x-t);
                for (t = zero, i = 2*(n+k); i >= m; i -= 2) t = one/(i/x-t);
                        a = t;
                        b = one;
                        /*
                         * estimate log((2/x)^n*n!) = n*log(2/x)+n*ln(n)
                         * hence, if n*(log(2n/x)) > ...
                         * single 8.8722839355e+01
                         * double 7.09782712893383973096e+02
                         * long double 1.1356523406294143949491931077970765006170
                         * then recurrent value may overflow and the result is
                         * likely underflow to zero
                         */
                        tmp = n;
                        v = two/x;
                        tmp = tmp*log(fabs(v*tmp));
                        if (tmp < 7.09782712893383973096e+02) {
                                for (i = n-1; i > 0; i--) {
                                        temp = b;
                                        b = ((i+i)/x)*b - a;
                                        a = temp;
                                }
                        } else {
                                for (i = n-1; i > 0; i--) {
                                        temp = b;
                                        b = ((i+i)/x)*b - a;
                                        a = temp;
                                        if (b > 1e100) {
                                                a /= b;
                                                t /= b;
                                                b  = 1.0;
                                        }
                                }
                        }
                        b = (t*j0(x)/b);
                }
```

```
        }
        if (sgn != 0)
        if (sgn == 1)
                return (-b);
        else
                return (b);
}

GENERIC
yn(int n, GENERIC x)
{
yn(int n, GENERIC x) {
        int i;
        int sign;
        GENERIC a, b, temp = 0, ox, on;

        ox = x;
        on = (GENERIC)n;
        ox = x; on = (GENERIC)n;
        if (isnan(x))
                return (x*x);   /* + -> * for Cheetah */
        if (x <= zero) {
                if (x == zero) {
                        /* return -one/zero; */
                        return (_SVID_libm_err((GENERIC)n, x, 12));
                } else {
                        /* return zero/zero; */
                        return (_SVID_libm_err((GENERIC)n, x, 13));
                }
        }
        if (!((int)_lib_version == libm_ieee ||
        if (!((int) _lib_version == libm_ieee ||
            (__xpg6 & _C99SUSv3_math_errexcept) != 0)) {
                if (x > X_TLOSS)
                        return (_SVID_libm_err(on, ox, 39));
        }
        sign = 1;
        if (n < 0) {
                n = -n;
                if ((n&1) == 1) sign = -1;
        }
        if (n == 0)
                return (y0(x));
        if (n == 1)
                return (sign*y1(x));
        if (!finite(x))
                return (zero);

        if (x > 1.0e91) {
                                /*
                                 * x >> n**2
                                 *  Jn(x) = cos(x-(2n+1)*pi/4)*sqrt(2/x*pi)
                                 *  Yn(x) = sin(x-(2n+1)*pi/4)*sqrt(2/x*pi)
                                 *  Let s = sin(x), c = cos(x),
                                 *  xn = x-(2n+1)*pi/4, sqt2 = sqrt(2), then
                                 *
                                 *    n sin(xn)*sqt2    cos(xn)*sqt2
                                 *      ----------------------------------
                                 *       0       s-c             c+s
                                 *       1      -s-c            -c+s
                                 *       2      -s+c            -c-s
                                 *       3       s+c             c-s
                                 */
                switch (n&3) {
                case 0:
                        temp =  sin(x)-cos(x);
                        break;
                case 1:
                        temp = -sin(x)-cos(x);
                        break;
                case 2:
                        temp = -sin(x)+cos(x);
                        break;
                case 3:
                        temp =  sin(x)+cos(x);
                        break;
                    case 0: temp =  sin(x)-cos(x); break;
                    case 1: temp = -sin(x)-cos(x); break;
                    case 2: temp = -sin(x)+cos(x); break;
                    case 3: temp =  sin(x)+cos(x); break;
                }
                b = invsqrtpi*temp/sqrt(x);
        } else {
                a = y0(x);
                b = y1(x);
                /*
                 * fix 1262058 and take care of non-default rounding
                 */
                for (i = 1; i < n; i++) {
                        temp = b;
                        b *= (GENERIC) (i + i) / x;
                        if (b <= -DBL_MAX)
                                break;
                        b -= a;
                        a = temp;
                }
        }
        if (sign > 0)
                return (b);
        else
                return (-b);
}
```

______unchanged_portion_omitted_

```
**********************************************************
    2507 Tue Jun  4 01:04:26 2019
new/usr/src/lib/libm/common/C/tanh.c
11175 libm should use signbit() correctly
11188 c99 math macros should return strictly backward compatible values
**********************************************************
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License (the "License").
 * You may not use this file except in compliance with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" replaced with your own identifying
 * information: Portions Copyright [yyyy] [name of copyright owner]
 *
 * CDDL HEADER END
 */

/*
 * Copyright 2011 Nexenta Systems, Inc.  All rights reserved.
 */
/*
 * Copyright 2006 Sun Microsystems, Inc.  All rights reserved.
 * Use is subject to license terms.
 */

#pragma weak __tanh = tanh

/* INDENT OFF */
/*
 * TANH(X)
 * RETURN THE HYPERBOLIC TANGENT OF X
 * code based on 4.3bsd
 * Modified by K.C. Ng for sun 4.0, Jan 31, 1987
 *
 * Method :
 *      1. reduce x to non-negative by tanh(-x) = - tanh(x).
 *      2.
 *          0      <  x <=  1.e-10 :  tanh(x) := x
 *                                                -expm1(-2x)
 *          1.e-10 <  x <=  1      :  tanh(x) := --------------
 *                                               expm1(-2x) + 2
 *                                                         2
 *          1      <= x <=  22.0   :  tanh(x) := 1 -  ---------------
 *                                                    expm1(2x) + 2
 *          22.0   <  x <= INF     :  tanh(x) := 1.
 *
 *      Note: 22 was chosen so that fl(1.0+2/(expm1(2*22)+2)) == 1.
 *
 * Special cases:
 *      tanh(NaN) is NaN;
 *      only tanh(0)=0 is exact for finite argument.
 */

#include "libm.h"
#include "libm_protos.h"
#include <math.h>
```

```
static const double
        one = 1.0,
        two = 2.0,
        small = 1.0e-10,
        big = 1.0e10;
/* INDENT ON */

double
tanh(double x)
{
tanh(double x) {
        double t, y, z;
        int signx;
        volatile double dummy __unused;

        if (isnan(x))
                return (x * x); /* + -> * for Cheetah */
        signx = signbit(x);
        t = fabs(x);
        z = one;
        if (t <= 22.0) {
                if (t > one)
                        z = one - two / (expm1(t + t) + two);
                else if (t > small) {
                        y = expm1(-t - t);
                        z = -y / (y + two);
                } else {
                        /* raise the INEXACT flag for non-zero t */
                        dummy = t + big;
#ifdef lint
                        dummy = dummy;
#endif
                        return (x);
                }
        } else if (!finite(t))
                return (copysign(1.0, x));
        else
                return ((signx != 0) ? -z + small * small : z - small * small);
                return (signx == 1 ? -z + small * small : z - small * small);

        return ((signx != 0) ? -z : z);
        return (signx == 1 ? -z : z);
}
______unchanged_portion_omitted_
```

```
**********************************************************
    1938 Tue Jun  4 01:04:27 2019
new/usr/src/lib/libm/common/LD/__rem_pio2l.c
11175 libm should use signbit() correctly
11188 c99 math macros should return strictly backward compatible values
**********************************************************
```

```
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License (the "License").
 * You may not use this file except in compliance with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" replaced with your own identifying
 * information: Portions Copyright [yyyy] [name of copyright owner]
 *
 * CDDL HEADER END
 */

/*
 * Copyright 2011 Nexenta Systems, Inc.  All rights reserved.
 */
/*
 * Copyright 2006 Sun Microsystems, Inc.  All rights reserved.
 * Use is subject to license terms.
 */

/*
 * __rem_pio2l(x,y)
/* __rem_pio2l(x,y)
 *
 * return the remainder of x rem pi/2 in y[0]+y[1]
 * by calling __rem_pio2m
 */

#include "libm.h"
#include "longdouble.h"

extern const int _TBL_ipio2l_inf[];

static const long double
    two24l = 16777216.0L,
    pio4   = 0.785398163397448309615660845819875721049 5L;

int
__rem_pio2l(long double x, long double *y)
{
        long double     z, w;
        double          t[3], v[5];
        int             e0, i, nx, n, sign;

        sign = signbitl(x);
        z = fabsl(x);
        if (z <= pio4) {
                y[0] = x;
                y[1] = 0;
                return (0);
        }
```

```
        e0 = ilogbl(z) - 23;
        z = scalbnl(z, -e0);
        for (i = 0; i < 3; i++) {
                t[i] = (double)((int)(z));
                z = (z - (long double)t[i]) * two24l;
        }
        nx = 3;
        while (t[nx-1] == 0.0)
                nx--;   /* omit trailing zeros */
        n = __rem_pio2m(t, v, e0, nx, 2, _TBL_ipio2l_inf);
        z = (long double)v[1];
        w = (long double)v[0];
        y[0] = z + w;
        y[1] = z - (y[0] - w);
        if (sign != 0) {
        if (sign == 1) {
                y[0] = -y[0];
                y[1] = -y[1];
                return (-n);
        }
        return (n);
}
______unchanged_portion_omitted_
```

```
**********************************************************
    6971 Tue Jun  4 01:04:27 2019
new/usr/src/lib/libm/common/LD/jnl.c
11175 libm should use signbit() correctly
11188 c99 math macros should return strictly backward compatible values
**********************************************************
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License (the "License").
 * You may not use this file except in compliance with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" replaced with your own identifying
 * information: Portions Copyright [yyyy] [name of copyright owner]
 *
 * CDDL HEADER END
 */

/*
 * Copyright 2011 Nexenta Systems, Inc.  All rights reserved.
 */
/*
 * Copyright 2006 Sun Microsystems, Inc.  All rights reserved.
 * Use is subject to license terms.
 */

#pragma weak __jnl = jnl
#pragma weak __ynl = ynl

/*
 * floating point Bessel's function of the 1st and 2nd kind
 * of order n: jn(n,x),yn(n,x);
 *
 * Special cases:
 *      y0(0)=y1(0)=yn(n,0) = -inf with division by zero signal;
 *      y0(-ve)=y1(-ve)=yn(n,-ve) are NaN with invalid signal.
 * Note 2. About jn(n,x), yn(n,x)
 *      For n=0, j0(x) is called,
 *      for n=1, j1(x) is called,
 *      for n<x, forward recursion us used starting
 *      from values of j0(x) and j1(x).
 *      for n>x, a continued fraction approximation to
 *      j(n,x)/j(n-1,x) is evaluated and then backward
 *      recursion is used starting from a supposed value
 *      for j(n,x). The resulting value of j(0,x) is
 *      compared with the actual value to correct the
 *      supposed value of j(n,x).
 *
 *      yn(n,x) is similar in all respects, except
 *      that forward recursion is used for all
 *      values of n>1.
 *
 */

#include "libm.h"
#include "longdouble.h"
#include <float.h>      /* LDBL_MAX */
```

```
#define GENERIC long double

static const GENERIC
invsqrtpi = 5.641895835477562869480794515607725858441e-0001L,
two  = 2.0L,
zero = 0.0L,
one  = 1.0L;

GENERIC
jnl(int n, GENERIC x)
{
jnl(n, x) int n; GENERIC x; {
        int i, sgn;
        GENERIC a, b, temp = 0, z, w;

        /*
         * J(-n,x) = (-1)^n * J(n, x), J(n, -x) = (-1)^n * J(n, x)
         * Thus, J(-n,x) = J(n,-x)
         */
        if (n < 0) {
                n = -n;
                x = -x;
        }
        if (n == 0)
                return (j0l(x));
        if (n == 1)
                return (j1l(x));
        if (x != x)
                return (x+x);
        if (n == 0) return (j0l(x));
        if (n == 1) return (j1l(x));
        if (x != x) return x+x;
        if ((n&1) == 0)
                sgn = 0;                        /* even n */
        else
                sgn = signbitl(x);      /* old n  */
        x = fabsl(x);
        if (x == zero || !finitel(x)) b = zero;
        else if ((GENERIC)n <= x) {
                        /*
                         * Safe to use
                         * J(n+1,x)=2n/x *J(n,x)-J(n-1,x)
                         */
                if (x > 1.0e91L) {
                                /*
                                 * x >> n**2
                                 *  Jn(x) = cos(x-(2n+1)*pi/4)*sqrt(2/x*pi)
                                 *  Yn(x) = sin(x-(2n+1)*pi/4)*sqrt(2/x*pi)
                                 *  Let s=sin(x), c=cos(x),
                                 *  xn=x-(2n+1)*pi/4, sqt2 = sqrt(2),then
                                 *
                                 *         n    sin(xn)*sqt2    cos(xn)*sqt2
                                 *      ----------------------------------
                                 *         0     s-c             c+s
                                 *         1    -s-c            -c+s
                                 *         2    -s+c            -c-s
                                 *         3     s+c             c-s
                                 */
                        switch (n&3) {
                        case 0:
                                temp =  cosl(x)+sinl(x);
                                break;
                        case 1:
                                temp = -cosl(x)+sinl(x);
                                break;
                        case 2:
                                temp = -cosl(x)-sinl(x);
                                break;
                        case 3:
                                temp =  cosl(x)-sinl(x);
                                break;
                    case 0: temp =  cosl(x)+sinl(x); break;
                    case 1: temp = -cosl(x)+sinl(x); break;
                    case 2: temp = -cosl(x)-sinl(x); break;
                    case 3: temp =  cosl(x)-sinl(x); break;
                        }
                        b = invsqrtpi*temp/sqrtl(x);
                } else {
                        a = j0l(x);
                        b = j1l(x);
                        for (i = 1; i < n; i++) {
                                temp = b;
                                /* avoid underflow */
                                b = b*((GENERIC)(i+i)/x) - a;
                    b = b*((GENERIC)(i+i)/x) - a; /* avoid underflow */
                                a = temp;
                        }
                }
        } else {
                if (x < 1e-17L) {       /* use J(n,x) = 1/n!*(x/2)^n */
                        b = powl(0.5L*x, (GENERIC)n);
                b = powl(0.5L*x, (GENERIC) n);
                        if (b != zero) {
                                for (a = one, i = 1; i <= n; i++)
                                        a *= (GENERIC)i;
                    for (a = one, i = 1; i <= n; i++) a *= (GENERIC)i;
                                b = b/a;
                        }
                } else {
                        /*
                         * use backward recurrence
                         *                      x      x^2      x^2
                         *  J(n,x)/J(n-1,x) =  ----   ------   ------   .....
                         *                      2n  - 2(n+1) - 2(n+2)
                         *
                         *                      1      1        1
                         *  (for large x)   =  ----  ------   ------   .....
                         *                      2n   2(n+1)   2(n+2)
                         *                      -- - ------ - ------ -
                         *                       x     x         x
                         *
                         * Let w = 2n/x and h=2/x, then the above quotient
                         * is equal to the continued fraction:
                         *                  1
                         *      = -----------------------
                         *                     1
                         *         w - -----------------
                         *                        1
                         *              w+h - ---------
                         *                     w+2h - ...
                         *
                         * To determine how many terms needed, let
                         * Q(0) = w, Q(1) = w(w+h) - 1,
                         * Q(k) = (w+k*h)*Q(k-1) - Q(k-2),
                         * When Q(k) > 1e4      good for single
                         * When Q(k) > 1e9      good for double
                         * When Q(k) > 1e17     good for quaduple
                         */
                        /* determine k */
            /* determin k */
                        GENERIC t, v;
```

```
                        double q0, q1, h, tmp;
                        int k, m;
                        w  = (n+n)/(double)x;
                        h = 2.0/(double)x;
                        q0 = w;
                        z = w+h;
                        q1 = w*z - 1.0;
                        k = 1;
                double q0, q1, h, tmp; int k, m;
                w  = (n+n)/(double)x; h = 2.0/(double)x;
                q0 = w;  z = w+h; q1 = w*z - 1.0; k = 1;
                        while (q1 < 1.0e17) {
                                k += 1;
                                z += h;
                        k += 1; z += h;
                                tmp = z*q1 - q0;
                                q0 = q1;
                                q1 = tmp;
                        }
                        m = n+n;
                        for (t = zero, i = 2*(n+k); i >= m; i -= 2)
                                t = one/(i/x-t);
                for (t = zero, i = 2*(n+k); i >= m; i -= 2) t = one/(i/x-t);
                        a = t;
                        b = one;
                        /*
                         * Estimate log((2/x)^n*n!) = n*log(2/x)+n*ln(n)
                         * hence, if n*(log(2n/x)) > ...
                         * single 8.8722839355e+01
                         * double 7.09782712893383973096e+02
                         * long double 1.135652340629414394949193107797076500617
                         * then recurrent value may overflow and the result is
                         * likely underflow to zero.
                         */
                        tmp = n;
                        v = two/x;
                        tmp = tmp*logl(fabsl(v*tmp));
                        if (tmp < 1.1356523406294143949491931077970765e+04L) {
                                for (i = n-1; i > 0; i--) {
                                        temp = b;
                                        b = ((i+i)/x)*b - a;
                                        a = temp;
                                }
                        } else {
                                for (i = n-1; i > 0; i--) {
                                        temp = b;
                                        b = ((i+i)/x)*b - a;
                                        a = temp;
                                        if (b > 1e1000L) {
                                                a /= b;
                                                t /= b;
                                                b  = 1.0;
                                        }
                                }
                        }
                        b = (t*j0l(x)/b);
                }
        }
        if (sgn != 0)
                return (-b);
        if (sgn == 1)
                return -b;
        else
                return (b);
                return b;
}
```

```
GENERIC
ynl(int n, GENERIC x)
{
ynl(n, x) int n; GENERIC x; {
        int i;
        int sign;
        GENERIC a, b, temp = 0;

        if (x != x)
                return (x+x);
                return x+x;
        if (x <= zero) {
                if (x == zero)
                        return (-one/zero);
                        return -one/zero;
                else
                        return (zero/zero);
                        return zero/zero;
        }
        sign = 1;
        if (n < 0) {
                n = -n;
                if ((n&1) == 1)
                        sign = -1;
                if ((n&1) == 1) sign = -1;
        }
        if (n == 0)
                return (y0l(x));
        if (n == 1)
                return (sign*y1l(x));
        if (!finitel(x))
                return (zero);
        if (n == 0) return (y0l(x));
        if (n == 1) return (sign*y1l(x));
        if (!finitel(x)) return zero;

        if (x > 1.0e91L) {
                                /*
                                 * x >> n**2
                                 * Jn(x) = cos(x-(2n+1)*pi/4)*sqrt(2/x*pi)
                                 *   Yn(x) = sin(x-(2n+1)*pi/4)*sqrt(2/x*pi)
                                 *   Let s=sin(x), c=cos(x),
                                 * xn=x-(2n+1)*pi/4, sqt2 = sqrt(2),then
                                 *
                                 *         n    sin(xn)*sqt2    cos(xn)*sqt2
                                 *      ----------------------------------
                                 *         0     s-c             c+s
                                 *         1    -s-c            -c+s
                                 *         2    -s+c            -c-s
                                 *         3     s+c             c-s
                                 */
                switch (n&3) {
                case 0:
                        temp =  sinl(x)-cosl(x);
                        break;
                case 1:
                        temp = -sinl(x)-cosl(x);
                        break;
                case 2:
                        temp = -sinl(x)+cosl(x);
                        break;
                case 3:
                        temp =  sinl(x)+cosl(x);
                        break;
                    case 0: temp =  sinl(x)-cosl(x); break;
```

```
                    case 1: temp = -sinl(x)-cosl(x); break;
                    case 2: temp = -sinl(x)+cosl(x); break;
                    case 3: temp =  sinl(x)+cosl(x); break;
                }
                b = invsqrtpi*temp/sqrtl(x);
        } else {
                a = y0l(x);
                b = y1l(x);
                /*
                 * fix 1262058 and take care of non-default rounding
                 */
                for (i = 1; i < n; i++) {
                        temp = b;
                        b *= (GENERIC) (i + i) / x;
                        if (b <= -LDBL_MAX)
                                break;
                        b -= a;
                        a = temp;
                }
        }
        if (sign > 0)
                return (b);
                return b;
        else
                return (-b);
                return -b;
}
______unchanged_portion_omitted_
```

```
**********************************************************
    2202 Tue Jun  4 01:04:27 2019
new/usr/src/lib/libm/common/Q/__rem_pio2l.c
11175 libm should use signbit() correctly
11188 c99 math macros should return strictly backward compatible values
**********************************************************
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License (the "License").
 * You may not use this file except in compliance with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" replaced with your own identifying
 * information: Portions Copyright [yyyy] [name of copyright owner]
 *
 * CDDL HEADER END
 */

/*
 * Copyright 2011 Nexenta Systems, Inc.  All rights reserved.
 */
/*
 * Copyright 2006 Sun Microsystems, Inc.  All rights reserved.
 * Use is subject to license terms.
 */

/*
 * __rem_pio2l(x,y)
 *
 * return the remainder of x rem pi/2 in y[0]+y[1] by calling __rem_pio2m
 */

#ifndef FDLIBM_BASED
#include "libm.h"
extern int __rem_pio2m(double *, double *, int, int, int, const int *);
#else                           /* FDLIBM_BASED */
#include "fdlibm.h"
#define __rem_pio2m     __kernel_rem_pio2
#endif                          /* FDLIBM_BASED */

#include "longdouble.h"

extern const int _TBL_ipio2l_inf[];

static const long double
        two24l = 16777216.0L,
        pio4   = 0.785398163397448309615660845819875721049L;

int
__rem_pio2l(long double x, long double *y)
{
__rem_pio2l(long double x, long double *y) {
        long double z, w;
        double t[5], v[5];
        int e0, i, nx, n, sign;
        const int *ipio2;
```

```
        sign = signbitl(x);
        z = fabsl(x);
        if (z <= pio4) {
                y[0] = x;
                y[1] = 0;
                return (0);
        }
        e0 = ilogbl(z) - 23;
        z = scalbnl(z, -e0);
        for (i = 0; i < 5; i++) {
                t[i] = (double)((int)(z));
                z = (z - (long double)t[i]) * two24l;
                t[i] = (double) ((int) (z));
                z = (z - (long double) t[i]) * two24l;
        }
        nx = 5;
        while (t[nx - 1] == 0.0)
                nx--;           /* skip zero term */
        ipio2 = _TBL_ipio2l_inf;
        n = __rem_pio2m(t, v, e0, nx, 3, (const int *)ipio2);
        z = (long double)v[2] + (long double)v[1];
        w = (long double)v[0];
        n = __rem_pio2m(t, v, e0, nx, 3, (const int *) ipio2);
        z = (long double) v[2] + (long double) v[1];
        w = (long double) v[0];
        y[0] = z + w;
        y[1] = z - (y[0] - w);
        if (sign != 0) {
        if (sign == 1) {
                y[0] = -y[0];
                y[1] = -y[1];
                return (-n);
        }
        return (n);
}
______unchanged_portion_omitted_
```

```
**********************************************************
    4215 Tue Jun  4 01:04:28 2019
new/usr/src/lib/libm/common/Q/atan2l.c
11175 libm should use signbit() correctly
11188 c99 math macros should return strictly backward compatible values
**********************************************************
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License (the "License").
 * You may not use this file except in compliance with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" replaced with your own identifying
 * information: Portions Copyright [yyyy] [name of copyright owner]
 *
 * CDDL HEADER END
 */

/*
 * Copyright 2011 Nexenta Systems, Inc.  All rights reserved.
 */
/*
 * Copyright 2006 Sun Microsystems, Inc.  All rights reserved.
 * Use is subject to license terms.
 */

/*
 * atan2l(y,x)
 *
 * Method :
 *      1. Reduce y to positive by atan2(y,x)=-atan2(-y,x).
 *      2. Reduce x to positive by (if x and y are unexceptional):
 *              ARG (x+iy) = arctan(y/x)           ... if x > 0,
 *              ARG (x+iy) = pi - arctan[y/(-x)]   ... if x < 0,
 *
 * Special cases:
 *
 *      ATAN2((anything), NaN ) is NaN;
 *      ATAN2(NAN , (anything) ) is NaN;
 *      ATAN2(+-0, +(anything but NaN)) is +-0  ;
 *      ATAN2(+-0, -(anything but NaN)) is +-PI ;
 *      ATAN2(+-(anything but 0 and NaN), 0) is +-PI/2;
 *      ATAN2(+-(anything but INF and NaN), +INF) is +-0 ;
 *      ATAN2(+-(anything but INF and NaN), -INF) is +-PI;
 *      ATAN2(+-INF,+INF ) is +-PI/4 ;
 *      ATAN2(+-INF,-INF ) is +-3PI/4;
 *      ATAN2(+-INF, (anything but,0,NaN, and INF)) is +-PI/2;
 *
 * Constants:
 * The hexadecimal values are the intended ones for the following constants.
 * The decimal values may be used, provided that the compiler will convert
 * from decimal to binary accurately enough to produce the hexadecimal values
 * shown.
 */

#pragma weak __atan2l = atan2l
```

```
#include "libm.h"
#include "longdouble.h"

static const long double
        zero    =  0.0L,
        tiny    =  1.0e-40L,
        one     =  1.0L,
        half    =  0.5L,
        PI3o4   =  2.356194490192344928846982537459627163148L,
        PIo4    =  0.785398163397448309615660845819875721049L,
        PIo2    =  1.570796326794896619231321691639751442099L,
        PI      =  3.141592653589793238462643383279502884197L,
        PI_lo   =  8.671810130123781024797044026043351968762e-35L;

long double
atan2l(long double y, long double x)
{
atan2l(long double y, long double x) {
        long double t, z;
        int k, m, signy, signx;

        if (x != x || y != y)
                return (x + y); /* return NaN if x or y is NAN */
        signy = signbitl(y);
        signx = signbitl(x);
        if (x == one)
                return (atanl(y));
        /* Ensure sign indicators are boolean */
        m = (signy != 0) + (signx != 0) + (signx != 0);
        m = signy + signx + signx;

        /* when y = 0 */
        if (y == zero)
                switch (m) {
                case 0:
                        return (y);     /* atan(+0,+anything) */
                case 1:
                        return (y);     /* atan(-0,+anything) */
                case 2:
                        return (PI + tiny);     /* atan(+0,-anything) */
                case 3:
                        return (-PI - tiny);    /* atan(-0,-anything) */
                }

        /* when x = 0 */
        if (x == zero)
                return (signy != 0 ? -PIo2 - tiny : PIo2 + tiny);
                return (signy == 1 ? -PIo2 - tiny : PIo2 + tiny);

        /* when x is INF */
        if (!finitel(x)) {
                if (!finitel(y)) {
                        switch (m) {
                        case 0:
                                return (PIo4 + tiny);   /* atan(+INF,+INF) */
                        case 1:
                                return (-PIo4 - tiny);  /* atan(-INF,+INF) */
                        case 2:
                                return (PI3o4 + tiny);  /* atan(+INF,-INF) */
                        case 3:
                                return (-PI3o4 - tiny); /* atan(-INF,-INF) */
                        }
                } else {
                        switch (m) {
                        case 0:
                                return (zero);  /* atan(+...,+INF) */
                        case 1:
                                return (-zero); /* atan(-...,+INF) */
                        case 2:
                                return (PI + tiny);     /* atan(+...,-INF) */
                        case 3:
                                return (-PI - tiny);    /* atan(-...,-INF) */
                        }
                }
        }
        /* when y is INF */
        if (!finitel(y))
                return (signy != 0 ? -PIo2 - tiny : PIo2 + tiny);
                return (signy == 1 ? -PIo2 - tiny : PIo2 + tiny);

        /* compute y/x */
        x = fabsl(x);
        y = fabsl(y);
        t = PI_lo;
        k = (ilogbl(y) - ilogbl(x));

        if (k > 120)
                z = PIo2 + half * t;
        else if (m > 1 && k < -120)
                z = zero;
        else
                z = atanl(y / x);

        switch (m) {
        case 0:
                return (z);     /* atan(+,+) */
        case 1:
                return (-z);    /* atan(-,+) */
        case 2:
                return (PI - (z - t));  /* atan(+,-) */
        case 3:
                return ((z - t) - PI);  /* atan(-,-) */
        }
        /* NOTREACHED */
        return (0.0L);
    return 0.0L;
}
```

______***unchanged_portion_omitted_***

```
**********************************************************
    6948 Tue Jun  4 01:04:28 2019
new/usr/src/lib/libm/common/Q/jnl.c
11175 libm should use signbit() correctly
11188 c99 math macros should return strictly backward compatible values
**********************************************************
```

```
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License (the "License").
 * You may not use this file except in compliance with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" replaced with your own identifying
 * information: Portions Copyright [yyyy] [name of copyright owner]
 *
 * CDDL HEADER END
 */

/*
 * Copyright 2011 Nexenta Systems, Inc.  All rights reserved.
 */
/*
 * Copyright 2006 Sun Microsystems, Inc.  All rights reserved.
 * Use is subject to license terms.
 */

#pragma weak __jnl = jnl
#pragma weak __ynl = ynl

/*
 * floating point Bessel's function of the 1st and 2nd kind
 * of order n: jn(n,x),yn(n,x);
 *
 * Special cases:
 *      y0(0)=y1(0)=yn(n,0) = -inf with division by zero signal;
 *      y0(-ve)=y1(-ve)=yn(n,-ve) are NaN with invalid signal.
 * Note 2. About jn(n,x), yn(n,x)
 *      For n=0, j0(x) is called,
 *      for n=1, j1(x) is called,
 *      for n<x, forward recursion us used starting
 *      from values of j0(x) and j1(x).
 *      for n>x, a continued fraction approximation to
 *      j(n,x)/j(n-1,x) is evaluated and then backward
 *      recursion is used starting from a supposed value
 *      for j(n,x). The resulting value of j(0,x) is
 *      compared with the actual value to correct the
 *      supposed value of j(n,x).
 *
 *      yn(n,x) is similar in all respects, except
 *      that forward recursion is used for all
 *      values of n>1.
 *
 */

#include "libm.h"
#include "longdouble.h"
#include <float.h>      /* LDBL_MAX */
```

```
#define GENERIC long double

static const GENERIC
invsqrtpi = 5.641895835477562869480794515607725858441e-0001L,
two  = 2.0L,
zero = 0.0L,
one  = 1.0L;

GENERIC
jnl(int n, GENERIC x)
{
jnl(n, x) int n; GENERIC x; {
        int i, sgn;
        GENERIC a, b, temp, z, w;

        /*
         * J(-n,x) = (-1)^n * J(n, x), J(n, -x) = (-1)^n * J(n, x)
         * Thus, J(-n,x) = J(n,-x)
         */
        if (n < 0) {
                n = -n;
                x = -x;
        }
        if (n == 0)
                return (j0l(x));
        if (n == 1)
                return (j1l(x));
        if (x != x)
                return (x+x);
        if ((n&1) == 0)
                sgn = 0;                        /* even n */
        else
                sgn = signbitl(x);      /* old n  */
        x = fabsl(x);
        if (x == zero || !finitel(x)) b = zero;
        else if ((GENERIC)n <= x) {
                                        /*
                                         * Safe to use
                                         * J(n+1,x)=2n/x *J(n,x)-J(n-1,x)
                                         */
                if (x > 1.0e91L) {
                                /*
                                 * x >> n**2
                                 *  Jn(x) = cos(x-(2n+1)*pi/4)*sqrt(2/x*pi)
                                 *   Yn(x) = sin(x-(2n+1)*pi/4)*sqrt(2/x*pi)
                                 *   Let s=sin(x), c=cos(x),
                                 *      xn=x-(2n+1)*pi/4, sqt2 = sqrt(2),then
                                 *
                                 *         n    sin(xn)*sqt2    cos(xn)*sqt2
                                 *      ----------------------------------
                                 *         0     s-c             c+s
                                 *         1    -s-c            -c+s
                                 *         2    -s+c            -c-s
                                 *         3     s+c             c-s
                                 */
                        switch (n&3) {
                        case 0:
                                temp =  cosl(x)+sinl(x);
                                break;
                        case 1:
                                temp = -cosl(x)+sinl(x);
                                break;
                        case 2:
                                temp = -cosl(x)-sinl(x);
                                break;
                        case 3:
                                temp =  cosl(x)-sinl(x);
                                break;
                    case 0: temp =  cosl(x)+sinl(x); break;
                    case 1: temp = -cosl(x)+sinl(x); break;
                    case 2: temp = -cosl(x)-sinl(x); break;
                    case 3: temp =  cosl(x)-sinl(x); break;
                        }
                        b = invsqrtpi*temp/sqrtl(x);
                } else {
                        a = j0l(x);
                        b = j1l(x);
                        for (i = 1; i < n; i++) {
                                temp = b;
                                /* avoid underflow */
                                b = b*((GENERIC)(i+i)/x) - a;
                    b = b*((GENERIC)(i+i)/x) - a; /* avoid underflow */
                                a = temp;
                        }
                }
        } else {
                if (x < 1e-17L) {       /* use J(n,x) = 1/n!*(x/2)^n */
                        b = powl(0.5L*x, (GENERIC)n);
                        if (b != zero) {
                                for (a = one, i = 1; i <= n; i++)
                                        a *= (GENERIC)i;
                    for (a = one, i = 1; i <= n; i++) a *= (GENERIC)i;
                                b = b/a;
                        }
                } else {
                        /* use backward recurrence */
                        /*
                         *                      x      x^2      x^2
                         *  J(n,x)/J(n-1,x) =  ----   ------   ------   .....
                         *                      2n  - 2(n+1) - 2(n+2)
                         *
                         *                      1      1        1
                         *  (for large x)   =  ----   ------   ------   .....
                         *                      2n   2(n+1)   2(n+2)
                         *                      -- - ------ - ------ -
                         *                       x     x         x
                         *
                         * Let w = 2n/x and h=2/x, then the above quotient
                         * is equal to the continued fraction:
                         *                  1
                         *      = -----------------------
                         *                     1
                         *         w - -----------------
                         *                        1
                         *              w+h - ---------
                         *                     w+2h - ...
                         *
                         * To determine how many terms needed, let
                         * Q(0) = w, Q(1) = w(w+h) - 1,
                         * Q(k) = (w+k*h)*Q(k-1) - Q(k-2),
                         * When Q(k) > 1e4      good for single
                         * When Q(k) > 1e9      good for double
                         * When Q(k) > 1e17     good for quaduple
                         */
                        /* determine k */
            /* determin k */
                        GENERIC t, v;
                        double q0, q1, h, tmp;
                        int k, m;
                        w  = (n+n)/(double)x;
                        h = 2.0/(double)x;
```

```
                        q0 = w;
                        z = w+h;
                        q1 = w*z - 1.0;
                        k = 1;
                double q0, q1, h, tmp; int k, m;
                w  = (n+n)/(double)x; h = 2.0/(double)x;
                q0 = w;  z = w+h; q1 = w*z - 1.0; k = 1;
                        while (q1 < 1.0e17) {
                                k += 1;
                                z += h;
                        k += 1; z += h;
                                tmp = z*q1 - q0;
                                q0 = q1;
                                q1 = tmp;
                        }
                        m = n+n;
                        for (t = zero, i = 2*(n+k); i >= m; i -= 2)
                                t = one/(i/x-t);
                for (t = zero, i = 2*(n+k); i >= m; i -= 2) t = one/(i/x-t);
                        a = t;
                        b = one;
                        /*
                         * estimate log((2/x)^n*n!) = n*log(2/x)+n*ln(n)
                         * hence, if n*(log(2n/x)) > ...
                         *  single 8.8722839355e+01
                         *  double 7.09782712893383973096e+02
                         *  long double 1.13565234062941439494919310779707650061
                         *  then recurrent value may overflow and the result is
                         *  likely underflow to zero
                         */
                        tmp = n;
                        v = two/x;
                        tmp = tmp*logl(fabsl(v*tmp));
                        if (tmp < 1.1356523406294143949491931077970765e+04L) {
                                for (i = n-1; i > 0; i--) {
                                        temp = b;
                                        b = ((i+i)/x)*b - a;
                                        a = temp;
                                }
                        } else {
                                for (i = n-1; i > 0; i--) {
                                        temp = b;
                                        b = ((i+i)/x)*b - a;
                                        a = temp;
                                        if (b > 1e1000L) {
                                                a /= b;
                                                t /= b;
                                                b  = 1.0;
                                        }
                                }
                        }
                        b = (t*j0l(x)/b);
                }
        }
        if (sgn != 0)
        if (sgn == 1)
                return (-b);
        else
                return (b);
}

GENERIC
ynl(int n, GENERIC x)
{
GENERIC ynl(n, x)
int n; GENERIC x; {
        int i;
        int sign;
        GENERIC a, b, temp;

        if (x != x)
                return (x+x);
        if (x <= zero) {
                if (x == zero)
                        return (-one/zero);
                else
                        return (zero/zero);
        }
        sign = 1;
        if (n < 0) {
                n = -n;
                if ((n&1) == 1) sign = -1;
        }
        if (n == 0)
                return (y0l(x));
        if (n == 1)
                return (sign*y1l(x));
        if (!finitel(x))
                return (zero);

        if (x > 1.0e91L) {
                /*
                 * x >> n**2
                 *   Jn(x) = cos(x-(2n+1)*pi/4)*sqrt(2/x*pi)
                 *   Yn(x) = sin(x-(2n+1)*pi/4)*sqrt(2/x*pi)
                 *   Let s = sin(x), c = cos(x),
                 *      xn = x-(2n+1)*pi/4, sqt2 = sqrt(2), then
                 *
                 *         n    sin(xn)*sqt2    cos(xn)*sqt2
                 *      ----------------------------------
                 *         0     s-c             c+s
                 *         1    -s-c            -c+s
                 *         2    -s+c            -c-s
                 *         3     s+c             c-s
        if (x > 1.0e91L) {      /* x >> n**2
                                    Jn(x) = cos(x-(2n+1)*pi/4)*sqrt(2/x*pi)
                                    Yn(x) = sin(x-(2n+1)*pi/4)*sqrt(2/x*pi)
                                    Let s = sin(x), c = cos(x),
                                        xn = x-(2n+1)*pi/4, sqt2 = sqrt(2), then

                                           n    sin(xn)*sqt2    cos(xn)*sqt2
                                        ----------------------------------
                                           0     s-c             c+s
                                           1    -s-c            -c+s
                                           2    -s+c            -c-s
                                           3     s+c             c-s
                 */
                switch (n&3) {
                case 0:
                        temp =  sinl(x)-cosl(x);
                        break;
                case 1:
                        temp = -sinl(x)-cosl(x);
                        break;
                case 2:
                        temp = -sinl(x)+cosl(x);
                        break;
                case 3:
                        temp =  sinl(x)+cosl(x);
                        break;
                    case 0: temp =  sinl(x)-cosl(x); break;
                    case 1: temp = -sinl(x)-cosl(x); break;
```

```
                    case 2: temp = -sinl(x)+cosl(x); break;
                    case 3: temp =  sinl(x)+cosl(x); break;
                }
                b = invsqrtpi*temp/sqrtl(x);
        } else {
                a = y0l(x);
                b = y1l(x);
                /*
                 * fix 1262058 and take care of non-default rounding
                 */
                for (i = 1; i < n; i++) {
                        temp = b;
                        b *= (GENERIC) (i + i) / x;
                        if (b <= -LDBL_MAX)
                                break;
                        b -= a;
                        a = temp;
                }
        }
        if (sign > 0)
                return (b);
        else
                return (-b);
}
______unchanged_portion_omitted_
```