```
**********************************************************
   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 */

/*
```

```
 122  * C99 7.12.14 comparison macros
 123  */
 124 #undef  isgreater
 125 #define isgreater(x, y)         __builtin_isgreater(x, y)
 126 #undef  isgreaterequal
 127 #define isgreaterequal(x, y)    __builtin_isgreaterequal(x, y)
 128 #undef  isless
 129 #define isless(x, y)            __builtin_isless(x, y)
 130 #undef  islessequal
 131 #define islessequal(x, y)       __builtin_islessequal(x, y)
 132 #undef  islessgreater
 133 #define islessgreater(x, y)     __builtin_islessgreater(x, y)
 134 #undef  isunordered
 135 #define isunordered(x, y)       __builtin_isunordered(x, y)
 136 #else   /* defined(__GNUC__) */
 137 #undef  HUGE_VAL
 138 #define HUGE_VAL        __builtin_huge_val
 139 #undef  HUGE_VALF
 140 #define HUGE_VALF       __builtin_huge_valf
 141 #undef  HUGE_VALL
 142 #define HUGE_VALL       __builtin_huge_vall
 143 #undef  INFINITY
 144 #define INFINITY        __builtin_infinity
 145 #undef  NAN
 146 #define NAN             __builtin_nan

 148 /*
 149  * C99 7.12.3 classification macros
 150  */
 151 #undef  fpclassify
 152 #define fpclassify(x)   __builtin_fpclassify(x)
 153 #undef  isfinite
 154 #define isfinite(x)     __builtin_isfinite(x)
 155 #undef  isinf
 156 #define isinf(x)        __builtin_isinf(x)
 157 #undef  isnan
 158 #define isnan(x)        __builtin_isnan(x)
 159 #undef  isnormal
 160 #define isnormal(x)     __builtin_isnormal(x)
 161 #undef  signbit
 162 #define signbit(x)      __builtin_signbit(x)

 164 /*
 165  * C99 7.12.14 comparison macros
 166  */
 167 #undef  isgreater
 168 #define isgreater(x, y)         ((x) __builtin_isgreater(y))
 169 #undef  isgreaterequal
 170 #define isgreaterequal(x, y)    ((x) __builtin_isgreaterequal(y))
 171 #undef  isless
 172 #define isless(x, y)            ((x) __builtin_isless(y))
 173 #undef  islessequal
 174 #define islessequal(x, y)       ((x) __builtin_islessequal(y))
 175 #undef  islessgreater
 176 #define islessgreater(x, y)     ((x) __builtin_islessgreater(y))
 177 #undef  isunordered
 178 #define isunordered(x, y)       ((x) __builtin_isunordered(y))
 179 #endif  /* defined(__GNUC__) */
 180 #endif  /* defined(_STDC_C99) || _XOPEN_SOURCE - 0 >= 600 || ... */

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



```
 188 #elif __FLT_EVAL_METHOD__ - 0 == 1
 189 typedef double float_t;
 190 typedef double double_t;
 191 #elif __FLT_EVAL_METHOD__ - 0 == 2
 192 typedef long double float_t;
 193 typedef long double double_t;
 194 #elif defined(__sparc) || defined(__amd64)
 195 typedef float float_t;
 196 typedef double double_t;
 197 #elif defined(__i386)
 198 typedef long double float_t;
 199 typedef long double double_t;
 200 #endif

 202 #undef  FP_ILOGB0
 203 #define FP_ILOGB0       (-2147483647)
 204 #undef  FP_ILOGBNAN
 205 #define FP_ILOGBNAN     2147483647

 207 #undef  MATH_ERRNO
 208 #define MATH_ERRNO      1
 209 #undef  MATH_ERREXCEPT
 210 #define MATH_ERREXCEPT  2
 211 #undef  math_errhandling
 212 #define math_errhandling        MATH_ERREXCEPT

 214 extern double acosh(double);
 215 extern double asinh(double);
 216 extern double atanh(double);

 218 extern double exp2(double);
 219 extern double expm1(double);
 220 extern int ilogb(double);
 221 extern double log1p(double);
 222 extern double log2(double);
 223 extern double logb(double);
 224 extern double scalbn(double, int);
 225 extern double scalbln(double, long int);

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

 230 extern double erf(double);
 231 extern double erfc(double);
 232 extern double lgamma(double);
 233 extern double tgamma(double);

 235 extern double nearbyint(double);
 236 extern double rint(double);
 237 extern long int lrint(double);
 238 extern double round(double);
 239 extern long int lround(double);
 240 extern double trunc(double);

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

 245 extern double copysign(double, double);
 246 extern double nan(const char *);
 247 extern double nextafter(double, double);
 248 extern double nexttoward(double, long double);

 250 extern double fdim(double, double);
 251 extern double fmax(double, double);
 252 extern double fmin(double, double);
```

```
254 extern double fma(double, double, double);

256 extern float acosf(float);
257 extern float asinf(float);
258 extern float atanf(float);
259 extern float atan2f(float, float);
260 extern float cosf(float);
261 extern float sinf(float);
262 extern float tanf(float);

264 extern float acoshf(float);
265 extern float asinhf(float);
266 extern float atanhf(float);
267 extern float coshf(float);
268 extern float sinhf(float);
269 extern float tanhf(float);

271 extern float expf(float);
272 extern float exp2f(float);
273 extern float expm1f(float);
274 extern float frexpf(float, int *);
275 extern int ilogbf(float);
276 extern float ldexpf(float, int);
277 extern float logf(float);
278 extern float log10f(float);
279 extern float log1pf(float);
280 extern float log2f(float);
281 extern float logbf(float);
282 extern float modff(float, float *);
283 extern float scalbnf(float, int);
284 extern float scalblnf(float, long int);

286 extern float cbrtf(float);
287 extern float fabsf(float);
288 extern float hypotf(float, float);
289 extern float powf(float, float);
290 extern float sqrtf(float);

292 extern float erff(float);
293 extern float erfcf(float);
294 extern float lgammaf(float);
295 extern float tgammaf(float);

297 extern float ceilf(float);
298 extern float floorf(float);
299 extern float nearbyintf(float);
300 extern float rintf(float);
301 extern long int lrintf(float);
302 extern float roundf(float);
303 extern long int lroundf(float);
304 extern float truncf(float);

306 extern float fmodf(float, float);
307 extern float remainderf(float, float);
308 extern float remquof(float, float, int *);

310 extern float copysignf(float, float);
311 extern float nanf(const char *);
312 extern float nextafterf(float, float);
313 extern float nexttowardf(float, long double);

315 extern float fdimf(float, float);
316 extern float fmaxf(float, float);
317 extern float fminf(float, float);

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



```
321 extern long double acosl(long double);
322 extern long double asinl(long double);
323 extern long double atanl(long double);
324 extern long double atan2l(long double, long double);
325 extern long double cosl(long double);
326 extern long double sinl(long double);
327 extern long double tanl(long double);

329 extern long double acoshl(long double);
330 extern long double asinhl(long double);
331 extern long double atanhl(long double);
332 extern long double coshl(long double);
333 extern long double sinhl(long double);
334 extern long double tanhl(long double);

336 extern long double expl(long double);
337 extern long double exp2l(long double);
338 extern long double expm1l(long double);
339 extern long double frexpl(long double, int *);
340 extern int ilogbl(long double);
341 extern long double ldexpl(long double, int);
342 extern long double logl(long double);
343 extern long double log10l(long double);
344 extern long double log1pl(long double);
345 extern long double log2l(long double);
346 extern long double logbl(long double);
347 extern long double modfl(long double, long double *);
348 extern long double scalbnl(long double, int);
349 extern long double scalblnl(long double, long int);

351 extern long double cbrtl(long double);
352 extern long double fabsl(long double);
353 extern long double hypotl(long double, long double);
354 extern long double powl(long double, long double);
355 extern long double sqrtl(long double);

357 extern long double erfl(long double);
358 extern long double erfcl(long double);
359 extern long double lgammal(long double);
360 extern long double tgammal(long double);

362 extern long double ceill(long double);
363 extern long double floorl(long double);
364 extern long double nearbyintl(long double);
365 extern long double rintl(long double);
366 extern long int lrintl(long double);
367 extern long double roundl(long double);
368 extern long int lroundl(long double);
369 extern long double truncl(long double);

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

375 extern long double copysignl(long double, long double);
376 extern long double nanl(const char *);
377 extern long double nextafterl(long double, long double);
378 extern long double nexttowardl(long double, long double);

380 extern long double fdiml(long double, long double);
381 extern long double fmaxl(long double, long double);
382 extern long double fminl(long double, long double);

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

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

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

394 extern long long int llrintl(long double);
395 extern long long int llroundl(long double);
396 #endif

398 #if !defined(__cplusplus)
399 #pragma does_not_read_global_data(asinh, exp2, expm1)
400 #pragma does_not_read_global_data(ilogb, log2)
401 #pragma does_not_read_global_data(scalbn, scalbln, cbrt)
402 #pragma does_not_read_global_data(erf, erfc, tgamma)
403 #pragma does_not_read_global_data(nearbyint, rint, lrint, round, lround, trunc)
404 #pragma does_not_read_global_data(remquo)
405 #pragma does_not_read_global_data(copysign, nan, nexttoward)
406 #pragma does_not_read_global_data(fdim, fmax, fmin, fma)
407 #pragma does_not_write_global_data(asinh, exp2, expm1)
408 #pragma does_not_write_global_data(ilogb, log2)
409 #pragma does_not_write_global_data(scalbn, scalbln, cbrt)
410 #pragma does_not_write_global_data(erf, erfc, tgamma)
411 #pragma does_not_write_global_data(nearbyint, rint, lrint, round, lround, trunc)
412 #pragma does_not_write_global_data(copysign, nan, nexttoward)
413 #pragma does_not_write_global_data(fdim, fmax, fmin, fma)

415 #pragma does_not_read_global_data(acosf, asinf, atanf, atan2f)
416 #pragma does_not_read_global_data(cosf, sinf, tanf)
417 #pragma does_not_read_global_data(acoshf, asinhf, atanhf, coshf, sinhf, tanhf)
418 #pragma does_not_read_global_data(expf, exp2f, expm1f, frexpf, ilogbf, ldexpf)
419 #pragma does_not_read_global_data(logf, log10f, log1pf, log2f, logbf)
420 #pragma does_not_read_global_data(modff, scalbnf, scalblnf)
421 #pragma does_not_read_global_data(cbrtf, fabsf, hypotf, powf, sqrtf)
422 #pragma does_not_read_global_data(erff, erfcf, lgammaf, tgammaf)
423 #pragma does_not_read_global_data(ceilf, floorf, nearbyintf)
424 #pragma does_not_read_global_data(rintf, lrintf, roundf, lroundf, truncf)
425 #pragma does_not_read_global_data(fmodf, remainderf, remquof)
426 #pragma does_not_read_global_data(copysignf, nanf, nextafterf, nexttowardf)
427 #pragma does_not_read_global_data(fdimf, fmaxf, fminf, fmaf)
428 #pragma does_not_write_global_data(acosf, asinf, atanf, atan2f)
429 #pragma does_not_write_global_data(cosf, sinf, tanf)
430 #pragma does_not_write_global_data(acoshf, asinhf, atanhf, coshf, sinhf, tanhf)
431 #pragma does_not_write_global_data(expf, exp2f, expm1f, ilogbf, ldexpf)
432 #pragma does_not_write_global_data(logf, log10f, log1pf, log2f, logbf)
433 #pragma does_not_write_global_data(cbrtf, fabsf, hypotf, powf, sqrtf)
434 #pragma does_not_write_global_data(erff, erfcf, tgammaf)
435 #pragma does_not_write_global_data(ceilf, floorf, nearbyintf)
436 #pragma does_not_write_global_data(rintf, lrintf, roundf, lroundf, truncf)
437 #pragma does_not_write_global_data(fmodf, remainderf)
438 #pragma does_not_write_global_data(copysignf, nanf, nextafterf, nexttowardf)
439 #pragma does_not_write_global_data(fdimf, fmaxf, fminf, fmaf)

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



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

467 #if !defined(_STRICT_STDC) && !defined(_NO_LONGLONG) || defined(_STDC_C99) || \
468         defined(__C99FEATURES__)
469 #pragma does_not_read_global_data(llrint, llround)
470 #pragma does_not_read_global_data(llrintf, llroundf, llrintl, llroundl)
471 #pragma does_not_write_global_data(llrint, llround)
472 #pragma does_not_write_global_data(llrintf, llroundf, llrintl, llroundl)
473 #endif
474 #endif  /* !defined(__cplusplus) */

476 #if defined(__MATHERR_ERRNO_DONTCARE)
477 #pragma does_not_read_global_data(acosh, atanh, hypot, lgamma, log1p, logb)
478 #pragma does_not_read_global_data(nextafter, remainder)
479 #pragma does_not_write_global_data(acosh, atanh, hypot, log1p, logb)
480 #pragma does_not_write_global_data(nextafter, remainder)

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

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

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

517 #if !defined(_STRICT_STDC) && !defined(_NO_LONGLONG) || defined(_STDC_C99) || \
```

```
 518         defined(__C99FEATURES__)
 519 #pragma no_side_effect(llrint, llround, llrintf, llroundf, llrintl, llroundl)
 520 #endif
 521 #endif  /* defined(__MATHERR_ERRNO_DONTCARE) */
 522 #endif  /* defined(__EXTENSIONS__) || defined(_STDC_C99) || ... */

 524 #ifdef __cplusplus
 525 }
```

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 = 5.64189583547756286948079451560772585844le-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
```

```
124                                  *         3     s+c             c-s
125                                  */
126                         switch (n&3) {
127                         case 0:
128                                 temp =  cos(x)+sin(x);
129                                 break;
130                         case 1:
131                                 temp = -cos(x)+sin(x);
132                                 break;
133                         case 2:
134                                 temp = -cos(x)-sin(x);
135                                 break;
136                         case 3:
137                                 temp =  cos(x)-sin(x);
138                                 break;
124                     case 0: temp =  cos(x)+sin(x); break;
125                     case 1: temp = -cos(x)+sin(x); break;
126                     case 2: temp = -cos(x)-sin(x); break;
127                     case 3: temp =  cos(x)-sin(x); break;
139                     }
140                     b = invsqrtpi*temp/sqrt(x);
141                 } else {
142                         a = j0(x);
143                         b = j1(x);
144                         for (i = 1; i < n; i++) {
145                                 temp = b;
146                                 /* avoid underflow */
147                                 b = b*((GENERIC)(i+i)/x) - a;
135                     b = b*((GENERIC)(i+i)/x) - a; /* avoid underflow */
148                                 a = temp;
149                         }
150                 }
151         } else {
152                 if (x < 1e-9) { /* use J(n,x) = 1/n!*(x/2)^n */
153                         b = pow(0.5*x, (GENERIC) n);
154                         if (b != zero) {
155                                 for (a = one, i = 1; i <= n; i++)
156                                         a *= (GENERIC)i;
143                     for (a = one, i = 1; i <= n; i++) a *= (GENERIC)i;
157                                 b = b/a;
158                         }
159                 } else {
160                         /*
161                          * use backward recurrence
162                          *                      x         x^2     x^2
163                          *  J(n,x)/J(n-1,x) =  ----   ------   ------   .....
164                          *                      2n  - 2(n+1) - 2(n+2)
165                          *
166                          *                      1         1         1
167                          *  (for large x)   =  ----  ------   ------   .....
168                          *                      2n   2(n+1)   2(n+2)
169                          *                      -- - ------ - ------ -
170                          *                       x       x             x
171                          *
172                          * Let w = 2n/x and h = 2/x, then the above quotient
173                          * is equal to the continued fraction:
174                          *                   1
175                          *      = -----------------------
176                          *                     1
177                          *         w - -----------------
178                          *                    1
179                          *                  w+h - ---------
180                          *                     w+2h - ...
181                          *
182                          * To determine how many terms needed, let
183                          * Q(0) = w, Q(1) = w(w+h) - 1,
```



```
184                          * Q(k) = (w+k*h)*Q(k-1) - Q(k-2),
185                          * When Q(k) > 1e4      good for single
186                          * When Q(k) > 1e9      good for double
187                          * When Q(k) > 1e17     good for quaduple
188                          */
189                         /* determine k */
176                 /* determin k */
190                         GENERIC t, v;
191                         double q0, q1, h, tmp;
192                         int k, m;
193                         w  = (n+n)/(double)x;
194                         h = 2.0/(double)x;
195                         q0 = w;
196                         z = w + h;
197                         q1 = w*z - 1.0;
198                         k = 1;

178                     double q0, q1, h, tmp; int k, m;
179                     w  = (n+n)/(double)x; h = 2.0/(double)x;
180                     q0 = w;  z = w + h; q1 = w*z - 1.0; k = 1;
200                         while (q1 < 1.0e9) {
201                                 k += 1;
202                                 z += h;
182                         k += 1; z += h;
203                                 tmp = z*q1 - q0;
204                                 q0 = q1;
205                                 q1 = tmp;
206                         }
207                         m = n+n;
208                         for (t = zero, i = 2*(n+k); i >= m; i -= 2)
209                                 t = one/(i/x-t);
188                     for (t = zero, i = 2*(n+k); i >= m; i -= 2) t = one/(i/x-t);
210                         a = t;
211                         b = one;
212                         /*
213                          * estimate log((2/x)^n*n!) = n*log(2/x)+n*ln(n)
214                          * hence, if n*(log(2n/x)) > ...
215                          * single 8.8722839355e+01
216                          * double 7.09782712893383973096e+02
217                          * long double 1.13565234062941439494919310779707650061 7
218                          * then recurrent value may overflow and the result is
219                          * likely underflow to zero
220                          */
221                         tmp = n;
222                         v = two/x;
223                         tmp = tmp*log(fabs(v*tmp));
224                         if (tmp < 7.09782712893383973096e+02) {
225                                 for (i = n-1; i > 0; i--) {
226                                         temp = b;
227                                         b = ((i+i)/x)*b - a;
228                                         a = temp;
229                                 }
230                         } else {
231                                 for (i = n-1; i > 0; i--) {
232                                         temp = b;
233                                         b = ((i+i)/x)*b - a;
234                                         a = temp;
235                                         if (b > 1e100) {
236                                                 a /= b;
237                                                 t /= b;
238                                                 b  = 1.0;
239                                         }
240                                 }
241                         }
242                         b = (t*j0(x)/b);
243                 }
```

```
244         }
245         if (sgn != 0)
224         if (sgn == 1)
246                 return (-b);
247         else
248                 return (b);
249 }

251 GENERIC
252 yn(int n, GENERIC x)
253 {
231 yn(int n, GENERIC x) {
254         int i;
255         int sign;
256         GENERIC a, b, temp = 0, ox, on;

258         ox = x;
259         on = (GENERIC)n;
236         ox = x; on = (GENERIC)n;
260         if (isnan(x))
261                 return (x*x);   /* + -> * for Cheetah */
262         if (x <= zero) {
263                 if (x == zero) {
264                         /* return -one/zero; */
265                         return (_SVID_libm_err((GENERIC)n, x, 12));
266                 } else {
267                         /* return zero/zero; */
268                         return (_SVID_libm_err((GENERIC)n, x, 13));
269                 }
270         }
271         if (!((int)_lib_version == libm_ieee ||
248         if (!((int) _lib_version == libm_ieee ||
272             (__xpg6 & _C99SUSv3_math_errexcept) != 0)) {
273                 if (x > X_TLOSS)
274                         return (_SVID_libm_err(on, ox, 39));
275         }
276         sign = 1;
277         if (n < 0) {
278                 n = -n;
279                 if ((n&1) == 1) sign = -1;
280         }
281         if (n == 0)
282                 return (y0(x));
283         if (n == 1)
284                 return (sign*y1(x));
285         if (!finite(x))
286                 return (zero);

288         if (x > 1.0e91) {
289                                 /*
290                                  * x >> n**2
291                                  *  Jn(x) = cos(x-(2n+1)*pi/4)*sqrt(2/x*pi)
292                                  *  Yn(x) = sin(x-(2n+1)*pi/4)*sqrt(2/x*pi)
293                                  *  Let s = sin(x), c = cos(x),
294                                  *  xn = x-(2n+1)*pi/4, sqt2 = sqrt(2), then
295                                  *
296                                  *    n sin(xn)*sqt2    cos(xn)*sqt2
297                                  *      ----------------------------------
298                                  *       0       s-c             c+s
299                                  *       1      -s-c            -c+s
300                                  *       2      -s+c            -c-s
301                                  *       3       s+c             c-s
302                                  */
303                 switch (n&3) {
304                 case 0:
305                         temp =  sin(x)-cos(x);
```



```
306                         break;
307                 case 1:
308                         temp = -sin(x)-cos(x);
309                         break;
310                 case 2:
311                         temp = -sin(x)+cos(x);
312                         break;
313                 case 3:
314                         temp =  sin(x)+cos(x);
315                         break;
281                     case 0: temp =  sin(x)-cos(x); break;
282                     case 1: temp = -sin(x)-cos(x); break;
283                     case 2: temp = -sin(x)+cos(x); break;
284                     case 3: temp =  sin(x)+cos(x); break;
316                 }
317                 b = invsqrtpi*temp/sqrt(x);
318         } else {
319                 a = y0(x);
320                 b = y1(x);
321                 /*
322                  * fix 1262058 and take care of non-default rounding
323                  */
324                 for (i = 1; i < n; i++) {
325                         temp = b;
326                         b *= (GENERIC) (i + i) / x;
327                         if (b <= -DBL_MAX)
328                                 break;
329                         b -= a;
330                         a = temp;
331                 }
332         }
333         if (sign > 0)
334                 return (b);
335         else
336                 return (-b);
337 }
```

_____***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   = 0.7853981633974483096156608458198757210495L;

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 = 5.64189583547756286948079451560772585844l-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;
```

```
123                         case 2:
124                                 temp = -cosl(x)-sinl(x);
125                                 break;
126                         case 3:
127                                 temp =  cosl(x)-sinl(x);
128                                 break;
113                     case 0: temp =  cosl(x)+sinl(x); break;
114                     case 1: temp = -cosl(x)+sinl(x); break;
115                     case 2: temp = -cosl(x)-sinl(x); break;
116                     case 3: temp =  cosl(x)-sinl(x); break;
129                         }
130                         b = invsqrtpi*temp/sqrtl(x);
131                 } else {
132                         a = j0l(x);
133                         b = j1l(x);
134                         for (i = 1; i < n; i++) {
135                                 temp = b;
136                                 /* avoid underflow */
137                                 b = b*((GENERIC)(i+i)/x) - a;
124                     b = b*((GENERIC)(i+i)/x) - a; /* avoid underflow */
138                                 a = temp;
139                         }
140                 }
141         } else {
142                 if (x < 1e-17L) {       /* use J(n,x) = 1/n!*(x/2)^n */
143                         b = powl(0.5L*x, (GENERIC)n);
130                 b = powl(0.5L*x, (GENERIC) n);
144                         if (b != zero) {
145                                 for (a = one, i = 1; i <= n; i++)
146                                     a *= (GENERIC)i;
132                     for (a = one, i = 1; i <= n; i++) a *= (GENERIC)i;
147                                 b = b/a;
148                         }
149                 } else {
150                         /*
151                          * use backward recurrence
152                          *                      x      x^2      x^2
153                          *  J(n,x)/J(n-1,x) =  ----   ------   ------   .....
154                          *                      2n  - 2(n+1) - 2(n+2)
155                          *
156                          *                      1      1        1
157                          *  (for large x)   =  ----  ------   ------   .....
158                          *                      2n   2(n+1)   2(n+2)
159                          *                      -- - ------ - ------ -
160                          *                       x     x         x
161                          *
162                          * Let w = 2n/x and h=2/x, then the above quotient
163                          * is equal to the continued fraction:
164                          *                   1
165                          *      = -----------------------
166                          *                  1
167                          *         w - -----------------
168                          *                     1
169                          *              w+h - ---------
170                          *                    w+2h - ...
171                          *
172                          * To determine how many terms needed, let
173                          * Q(0) = w, Q(1) = w(w+h) - 1,
174                          * Q(k) = (w+k*h)*Q(k-1) - Q(k-2),
175                          * When Q(k) > 1e4      good for single
176                          * When Q(k) > 1e9      good for double
177                          * When Q(k) > 1e17     good for quaduple
178                          */
179                         /* determine k */
165                 /* determin k */
180                         GENERIC t, v;
```



```
181                         double q0, q1, h, tmp;
182                         int k, m;
183                         w  = (n+n)/(double)x;
184                         h = 2.0/(double)x;
185                         q0 = w;
186                         z = w+h;
187                         q1 = w*z - 1.0;
188                         k = 1;
167                 double q0, q1, h, tmp; int k, m;
168                 w  = (n+n)/(double)x; h = 2.0/(double)x;
169                 q0 = w;  z = w+h; q1 = w*z - 1.0; k = 1;
189                         while (q1 < 1.0e17) {
190                                 k += 1;
191                                 z += h;
171                         k += 1; z += h;
192                                 tmp = z*q1 - q0;
193                                 q0 = q1;
194                                 q1 = tmp;
195                         }
196                         m = n+n;
197                         for (t = zero, i = 2*(n+k); i >= m; i -= 2)
198                                 t = one/(i/x-t);
177                 for (t = zero, i = 2*(n+k); i >= m; i -= 2) t = one/(i/x-t);
199                         a = t;
200                         b = one;
201                         /*
202                          * Estimate log((2/x)^n*n!) = n*log(2/x)+n*ln(n)
203                          * hence, if n*(log(2n/x)) > ...
204                          * single 8.8722839355e+01
205                          * double 7.09782712893383973096e+02
206                          * long double 1.13565234062941439494919310779707650061 7
207                          * then recurrent value may overflow and the result is
208                          * likely underflow to zero.
209                          */
210                         tmp = n;
211                         v = two/x;
212                         tmp = tmp*logl(fabsl(v*tmp));
213                         if (tmp < 1.13565234062941439494919310779707650e+04L) {
214                                 for (i = n-1; i > 0; i--) {
215                                         temp = b;
216                                         b = ((i+i)/x)*b - a;
217                                         a = temp;
218                                 }
219                         } else {
220                                 for (i = n-1; i > 0; i--) {
221                                         temp = b;
222                                         b = ((i+i)/x)*b - a;
223                                         a = temp;
224                                         if (b > 1e1000L) {
225                                                 a /= b;
226                                                 t /= b;
227                                                 b  = 1.0;
228                                         }
229                                 }
230                         }
231                         b = (t*j0l(x)/b);
232                 }
233         }
234         if (sgn != 0)
235                 return (-b);
213         if (sgn == 1)
214                 return -b;
236         else
237                 return (b);
216                 return b;
238 }
```

```
240 GENERIC
241 ynl(int n, GENERIC x)
242 {
220 ynl(n, x) int n; GENERIC x; {
243         int i;
244         int sign;
245         GENERIC a, b, temp = 0;

247         if (x != x)
248                 return (x+x);
226                 return x+x;
249         if (x <= zero) {
250                 if (x == zero)
251                         return (-one/zero);
229                         return -one/zero;
252                 else
253                         return (zero/zero);
231                         return zero/zero;
254         }
255         sign = 1;
256         if (n < 0) {
257                 n = -n;
258                 if ((n&1) == 1)
259                         sign = -1;
236                 if ((n&1) == 1) sign = -1;
260         }
261         if (n == 0)
262                 return (y0l(x));
263         if (n == 1)
264                 return (sign*y1l(x));
265         if (!finitel(x))
266                 return (zero);
238         if (n == 0) return (y0l(x));
239         if (n == 1) return (sign*y1l(x));
240         if (!finitel(x)) return zero;

268         if (x > 1.0e91L) {
269                                 /*
270                                  * x >> n**2
271                                  * Jn(x) = cos(x-(2n+1)*pi/4)*sqrt(2/x*pi)
272                                  *   Yn(x) = sin(x-(2n+1)*pi/4)*sqrt(2/x*pi)
273                                  *   Let s=sin(x), c=cos(x),
274                                  * xn=x-(2n+1)*pi/4, sqt2 = sqrt(2),then
275                                  *
276                                  *         n    sin(xn)*sqt2    cos(xn)*sqt2
277                                  *      ----------------------------------
278                                  *         0     s-c             c+s
279                                  *         1    -s-c            -c+s
280                                  *         2    -s+c            -c-s
281                                  *         3     s+c             c-s
282                                  */
283                 switch (n&3) {
284                 case 0:
285                         temp =  sinl(x)-cosl(x);
286                         break;
287                 case 1:
288                         temp = -sinl(x)-cosl(x);
289                         break;
290                 case 2:
291                         temp = -sinl(x)+cosl(x);
292                         break;
293                 case 3:
294                         temp =  sinl(x)+cosl(x);
295                         break;
258                     case 0: temp =  sinl(x)-cosl(x); break;
```



```
259                     case 1: temp = -sinl(x)-cosl(x); break;
260                     case 2: temp = -sinl(x)+cosl(x); break;
261                     case 3: temp =  sinl(x)+cosl(x); break;
296                 }
297                 b = invsqrtpi*temp/sqrtl(x);
298         } else {
299                 a = y0l(x);
300                 b = y1l(x);
301                 /*
302                  * fix 1262058 and take care of non-default rounding
303                  */
304                 for (i = 1; i < n; i++) {
305                         temp = b;
306                         b *= (GENERIC) (i + i) / x;
307                         if (b <= -LDBL_MAX)
308                                 break;
309                         b -= a;
310                         a = temp;
311                 }
312         }
313         if (sign > 0)
314                 return (b);
280                 return b;
315         else
316                 return (-b);
282                 return -b;
317 }
______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   = 0.78539816339744830961566084581987572104 95L;

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   =  2.3561944901923449288469825374596271631 48L,
        PIo4    =  0.7853981633974483096156608458198757210 49L,
        PIo2    =  1.5707963267948966192313216916397514420 99L,
        PI      =  3.1415926535897932384626433832795028841 97L,
        PI_lo   =  8.6718101301237810247970440260433519687 62e-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) */
```

```
124                         case 1:
125                                 return (-zero); /* atan(-...,+INF) */
126                         case 2:
127                                 return (PI + tiny);     /* atan(+...,-INF) */
128                         case 3:
129                                 return (-PI - tiny);    /* atan(-...,-INF) */
130                         }
131                 }
132         }
133         /* when y is INF */
134         if (!finitel(y))
135                 return (signy != 0 ? -PIo2 - tiny : PIo2 + tiny);
133                 return (signy == 1 ? -PIo2 - tiny : PIo2 + tiny);

137         /* compute y/x */
138         x = fabsl(x);
139         y = fabsl(y);
140         t = PI_lo;
141         k = (ilogbl(y) - ilogbl(x));

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

150         switch (m) {
151         case 0:
152                 return (z);     /* atan(+,+) */
153         case 1:
154                 return (-z);    /* atan(-,+) */
155         case 2:
156                 return (PI - (z - t));  /* atan(+,-) */
157         case 3:
158                 return ((z - t) - PI);  /* atan(-,-) */
159         }
160         /* NOTREACHED */
161         return (0.0L);
159     return 0.0L;
162 }
```

_____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 = 5.64189583547756286948079451560772585844l-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;
```

```
126                         case 3:
127                                 temp =  cosl(x)-sinl(x);
128                                 break;
116                     case 0: temp =  cosl(x)+sinl(x); break;
117                     case 1: temp = -cosl(x)+sinl(x); break;
118                     case 2: temp = -cosl(x)-sinl(x); break;
119                     case 3: temp =  cosl(x)-sinl(x); break;
129                         }
130                         b = invsqrtpi*temp/sqrtl(x);
131                 } else {
132                         a = j0l(x);
133                         b = j1l(x);
134                         for (i = 1; i < n; i++) {
135                                 temp = b;
136                                 /* avoid underflow */
137                                 b = b*((GENERIC)(i+i)/x) - a;
127                     b = b*((GENERIC)(i+i)/x) - a; /* avoid underflow */
138                                 a = temp;
139                         }
140                 }
141         } else {
142                 if (x < 1e-17L) {       /* use J(n,x) = 1/n!*(x/2)^n */
143                         b = powl(0.5L*x, (GENERIC)n);
144                         if (b != zero) {
145                                 for (a = one, i = 1; i <= n; i++)
146                                         a *= (GENERIC)i;
135                     for (a = one, i = 1; i <= n; i++) a *= (GENERIC)i;
147                                 b = b/a;
148                         }
149                 } else {
150                         /* use backward recurrence */
151                         /*
152                          *                          x      x^2      x^2
153                          *  J(n,x)/J(n-1,x) =  ----   ------   ------   .....
154                          *                          2n  - 2(n+1) - 2(n+2)
155                          *
156                          *                          1      1        1
157                          *  (for large x)   =  ----  ------   ------   .....
158                          *                          2n   2(n+1)   2(n+2)
159                          *                          -- - ------ - ------ -
160                          *                           x     x         x
161                          *
162                          * Let w = 2n/x and h=2/x, then the above quotient
163                          * is equal to the continued fraction:
164                          *                      1
165                          *      = -----------------------
166                          *                     1
167                          *         w - -----------------
168                          *                       1
169                          *              w+h - ---------
170                          *                    w+2h - ...
171                          *
172                          * To determine how many terms needed, let
173                          * Q(0) = w, Q(1) = w(w+h) - 1,
174                          * Q(k) = (w+k*h)*Q(k-1) - Q(k-2),
175                          * When Q(k) > 1e4      good for single
176                          * When Q(k) > 1e9      good for double
177                          * When Q(k) > 1e17     good for quaduple
178                          */
179                         /* determine k */
168                 /* determin k */
180                         GENERIC t, v;
181                         double q0, q1, h, tmp;
182                         int k, m;
183                         w  = (n+n)/(double)x;
184                         h = 2.0/(double)x;
```



```
185                         q0 = w;
186                         z = w+h;
187                         q1 = w*z - 1.0;
188                         k = 1;
170                 double q0, q1, h, tmp; int k, m;
171                 w  = (n+n)/(double)x; h = 2.0/(double)x;
172                 q0 = w;  z = w+h; q1 = w*z - 1.0; k = 1;
189                         while (q1 < 1.0e17) {
190                                 k += 1;
191                                 z += h;
174                         k += 1; z += h;
192                                 tmp = z*q1 - q0;
193                                 q0 = q1;
194                                 q1 = tmp;
195                         }
196                         m = n+n;
197                         for (t = zero, i = 2*(n+k); i >= m; i -= 2)
198                                 t = one/(i/x-t);
180                 for (t = zero, i = 2*(n+k); i >= m; i -= 2) t = one/(i/x-t);
199                         a = t;
200                         b = one;
201                         /*
202                          * estimate log((2/x)^n*n!) = n*log(2/x)+n*ln(n)
203                          * hence, if n*(log(2n/x)) > ...
204                          *  single 8.8722839355e+01
205                          *  double 7.09782712893383973096e+02
206                          *  long double 1.13565234062941439494919310779707650061
207                          *  then recurrent value may overflow and the result is
208                          *  likely underflow to zero
209                          */
210                         tmp = n;
211                         v = two/x;
212                         tmp = tmp*logl(fabsl(v*tmp));
213                         if (tmp < 1.13565234062941439494919310779707650e+04L) {
214                                 for (i = n-1; i > 0; i--) {
215                                         temp = b;
216                                         b = ((i+i)/x)*b - a;
217                                         a = temp;
218                                 }
219                         } else {
220                                 for (i = n-1; i > 0; i--) {
221                                         temp = b;
222                                         b = ((i+i)/x)*b - a;
223                                         a = temp;
224                                         if (b > 1e1000L) {
225                                                 a /= b;
226                                                 t /= b;
227                                                 b  = 1.0;
228                                         }
229                                 }
230                         }
231                         b = (t*j0l(x)/b);
232                 }
233         }
234         if (sgn != 0)
216         if (sgn == 1)
235                 return (-b);
236         else
237                 return (b);
238 }

240 GENERIC
241 ynl(int n, GENERIC x)
242 {
222 GENERIC ynl(n, x)
223 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_*