```c
/* @(#)e_log.c 5.1 93/09/24 */
/*
 * ====================================================
 * Copyright (C) 1993 by Sun Microsystems, Inc. All rights reserved.
 *
 * Developed at SunPro, a Sun Microsystems, Inc. business.
 * Permission to use, copy, modify, and distribute this
 * software is freely granted, provided that this notice
 * is preserved.
 * ====================================================
 */

#if defined(LIBM_SCCS) && !defined(lint)
static char rcsid[] = "$NetBSD: e_log.c,v 1.8 1995/05/10 20:45:49 jtc Exp $";
#endif

/* __ieee754_log(x)
 * Return the logarithm of x
 *
 * Method :
 *   1. Argument Reduction: find k and f such that
 *                      x = 2^k * (1+f),
 *         where  sqrt(2)/2 < 1+f < sqrt(2) .
 *
 *   2. Approximation of log(1+f).
 *      Let s = f/(2+f) ; based on log(1+f) = log(1+s) - log(1-s)
 *               = 2s + 2/3 s**3 + 2/5 s**5 + .....,
 *               = 2s + s*R
 *      We use a special Reme algorithm on [0,0.1716] to generate
 *      a polynomial of degree 14 to approximate R The maximum error
 *      of this polynomial approximation is bounded by 2**-58.45. In
 *      other words,
 *                      2      4      6      8      10      12      14
 *          R(z) ~ Lg1*s +Lg2*s +Lg3*s +Lg4*s +Lg5*s  +Lg6*s  +Lg7*s
 *      (the values of Lg1 to Lg7 are listed in the program)
 *      and
 *          |      2          14          |     -58.45
 *          | Lg1*s +...+Lg7*s    -  R(z) | <= 2
 *          |                             |
 *      Note that 2s = f - s*f = f - hfsq + s*hfsq, where hfsq = f*f/2.
 *      In order to guarantee error in log below 1ulp, we compute log
 *      by
 *              log(1+f) = f - s*(f - R)        (if f is not too large)
 *              log(1+f) = f - (hfsq - s*(hfsq+R)).     (better accuracy)
 *
 *      3. Finally,  log(x) = k*ln2 + log(1+f).
 *                          = k*ln2_hi+(f-(hfsq-(s*(hfsq+R)+k*ln2_lo)))
 *         Here ln2 is split into two floating point number:
 *                      ln2_hi + ln2_lo,
 *         where n*ln2_hi is always exact for |n| < 2000.
 *
 * Special cases:
 *      log(x) is NaN with signal if x < 0 (including -INF) ;
 *      log(+INF) is +INF; log(0) is -INF with signal;
 *      log(NaN) is that NaN with no signal.
 *
 * Accuracy:
 *      according to an error analysis, the error is always less than
 *      1 ulp (unit in the last place).
 *
 * Constants:
 * The hexadecimal values are the intended ones for the following
```



```c
 * 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.
 */

#include "math.h"
#include "math_private.h"

#ifdef __STDC__
static const double
#else
static double
#endif
ln2_hi  =  6.93147180369123816490e-01,  /* 3fe62e42 fee00000 */
ln2_lo  =  1.90821492927058770002e-10,  /* 3dea39ef 35793c76 */
two54   =  1.80143985094819840000e+16,  /* 43500000 00000000 */
Lg1 = 6.666666666666735130e-01,  /* 3FE55555 55555593 */
Lg2 = 3.999999999940941908e-01,  /* 3FD99999 9997FA04 */
Lg3 = 2.857142874366239149e-01,  /* 3FD24924 94229359 */
Lg4 = 2.222219843214978396e-01,  /* 3FCC71C5 1D8E78AF */
Lg5 = 1.818357216161805012e-01,  /* 3FC74664 96CB03DE */
Lg6 = 1.531383769920937332e-01,  /* 3FC39A09 D078C69F */
Lg7 = 1.479819860511658591e-01;  /* 3FC2F112 DF3E5244 */

#ifdef __STDC__
static const double zero   =  0.0;
#else
static double zero   =  0.0;
#endif

#ifdef __STDC__
        double __ieee754_log(double x)
#else
        double __ieee754_log(x)
        double x;
#endif
{
        double hfsq,f,s,z,R,w,t1,t2,dk;
        int32_t k,hx,i,j;
        u_int32_t lx;

        EXTRACT_WORDS(hx,lx,x);

        k=0;
        if (hx < 0x00100000) {                  /* x < 2**-1022  */
            if (((hx&0x7fffffff)|lx)==0)
                return -two54/zero;             /* log(+-0)=-inf */
            if (hx<0) return (x-x)/zero;        /* log(-#) = NaN */
            k -= 54; x *= two54; /* subnormal number, scale up x */
            GET_HIGH_WORD(hx,x);
        }
        if (hx >= 0x7ff00000) return x+x;
        k += (hx>>20)-1023;
        hx &= 0x000fffff;
        i = (hx+0x95f64)&0x100000;
        SET_HIGH_WORD(x,hx|(i^0x3ff00000));     /* normalize x or x/2 */
        k += (i>>20);
        f = x-1.0;
        if((0x000fffff&(2+hx))<3) {     /* |f| < 2**-20 */
            if(f==zero) if(k==0) return zero;  else {dk=(double)k;
                                 return dk*ln2_hi+dk*ln2_lo;}
            R = f*f*(0.5-0.33333333333333333*f);
```

```
            if(k==0) return f-R; else {dk=(double)k;
                     return dk*ln2_hi-((R-dk*ln2_lo)-f);}
        }
        s = f/(2.0+f);
        dk = (double)k;
        z = s*s;
        i = hx-0x6147a;
        w = z*z;
        j = 0x6b851-hx;
        t1= w*(Lg2+w*(Lg4+w*Lg6));
        t2= z*(Lg1+w*(Lg3+w*(Lg5+w*Lg7)));
        i |= j;
        R = t2+t1;
        if(i>0) {
            hfsq=0.5*f*f;
            if(k==0) return f-(hfsq-s*(hfsq+R)); else
                     return dk*ln2_hi-((hfsq-(s*(hfsq+R)+dk*ln2_lo))-f);
        } else {
            if(k==0) return f-s*(f-R); else
                     return dk*ln2_hi-((s*(f-R)-dk*ln2_lo)-f);
        }
}
```

```
/* Vizsgáljuk, hogy a (PI/2)^n/n! mikor csökken 10^(-k) alá. */

#define PI 3.141592654

double hatvany(double a,int n)
{
    double v=1;
    while (--n>=0)
        v*=a;
    return v;
}

double faktorialis(int n)
{
    double v=1;
    while (n>1)
        v*=n--;
    return v;
}

main()
{
    double a;
    int n;
    for (n=1;n<=25;++n)
        printf("(PI/2)^%d/%d!=%20.18f\n",n,n,hatvany(PI/2,n)/faktorialis(n));
}
```

```c
/* Vizsgáljuk, hogy a (PI/2)^n/n! mikor csökken 10^(-k) alá. */

#define PI 3.141592654

double hatvany(double a,int n)
{
    double v=1;
    while (--n>=0)
        v*=a;
    return v;
}

long faktorialis(int n)
{
    long v=1;
    while (n>1)
        v*=n--;
    return v;
}

main()
{
    double a;
    int n;
    for (n=1;n<=20;++n)
        printf("(PI/2)^%d/%d!=%10.8f\n",n,n,hatvany(PI/2,n)/faktorialis(n));
}
```

```
/* Vizsgáljuk, hogy a (PI/2)^n/n! mikor csökken 10^(-k) alá. */

#define PI 3.141592654

double hatvany(double a,int n)
{
    double v=1;
    while (--n>=0)
        v*=a;
    return v;
}

long faktorialis(int n)
{
    long v=1;
    while (n>1)
        v*=n--;
    return v;
}

main()
{
    double a;
    int n;
    for (n=1;n<=20;++n)
        printf("%d!=%d, (PI/2)^%d/%d!=%10.8f\n",n,faktorialis(n),
            n,n,hatvany(PI/2,n)/faktorialis(n));
}
```

```
main()
{
 double s=0;
 long n;
 for (n=1;n<100;++n)
  s+=1/(double)(n*n);
 printf("%8.6f\n",s);
}
```

```
int prim_e(int a)
{
 int v=1;
 long i;
 if (a<2)
  v=0;
 for (i=2;i*i<=a;i+=1)
  if (a%i==0) v=0;
 return v;
}

main()
{
 double t=0;
 long n;
 for (n=1;n<10000;n+=1)
  if (prim_e(n))
   t+=1/(double)(n*n);
 printf("%8.6f\n",t);
}
```