MathLib Header Files

This appendix shows the contents of the two MathLib header files `fp.h` and `fenv.h`. You can use this appendix to see where and how a MathLib function is defined and to see which transcendental functions are available in MathLib.

Floating-Point Header File (fp.h)

The header file `fp.h` defines a collection of numerical functions designed to facilitate a wide range of numerical programming. It is modeled after the FPCE technical report. This file declares many functions in support of numerical programming. It provides a superset of `math.h` and `sane.h` functions. Some functionality previously found in `sane.h` on 680x0-based Macintosh computers and not in the FPCE `fp.h` can be found in this `fp.h` under the heading `__NOEXTENSIONS__`.

Constants

```
#ifndef  __FP__
#define  __FP__

/* efficient types are included in Types.h. */
#ifndef  __TYPES__
#include <Types.h>
#endif

#ifdef   powerc
#define          LONG_DOUBLE_SIZE     16
#elif    mc68881
#define          LONG_DOUBLE_SIZE     12
#else
#define          LONG_DOUBLE_SIZE     10
#endif      /* powerc */

#define          DOUBLE_SIZE           8

#define          HUGE_VAL             __inf()
#define          INFINITY             __inf()
#define          NAN                  nan("255")
```

```
/* the macro DECIMAL_DIG is obtained by satisfying the constraint that the
   conversion from double to decimal and back is the identity function. */

#ifdef   powerc
#define          DECIMAL_DIG         36
#else
#define          DECIMAL_DIG         21
#endif   /* powerc */

#define      SIGDIGLEN      36           /* significant decimal digits */
#define      DECSTROUTLEN   80           /* max length for dec2str output */
#define      FLOATDECIMAL   ((char)(0))
#define      FIXEDDECIMAL   ((char)(1))
```

Inquiry Macros

```
#define  fpclassify  (x)   ((    sizeof (x) == LONG_DOUBLE_SIZE)     ?  \
                                 __fpclassify  (x)                  :  \
                                 (sizeof (x) == DOUBLE_SIZE)        ?  \
                                 __fpclassifyd (x)                  :  \
                                 __fpclassifyf (x))

/* isnormal is nonzero if and only if the argument x is normalized. */

#define  isnormal    (x)   ((    sizeof (x) == LONG_DOUBLE_SIZE)     ?  \
                                 __isnormal (x)                     :  \
                                 (sizeof (x) == DOUBLE_SIZE)        ?  \
                                 __isnormald (x)                    :  \
                                 __isnormalf (x))

/* isfinite is nonzero if and only if the argument x is finite. */

#define  isfinite    (x)   ((    sizeof (x) == LONG_DOUBLE_SIZE)     ?  \
                                 __isfinite (x)                     :  \
                                 ( sizeof (x) == DOUBLE_SIZE)       ?  \
                                 __isfinited (x)                    :  \
                                 __isfinitef (x))

/* isnan is nonzero if and only if the argument x is a NaN. */
```

```
#define  isnan       (x)   ((    sizeof (x) == LONG_DOUBLE_SIZE)     ?  \
                                 __isnan (x)                         :  \
                                 (sizeof (x) == DOUBLE_SIZE)         ?  \
                                 __isnand (x)                        :  \
                                 __isnanf (x))

/* signbit is nonzero if and only if the sign of the argument x is
   negative. This includes NaNs, infinities and zeros. */

#define  signbit     (x)   ((    sizeof (x) == LONG_DOUBLE_SIZE)     ?  \
                                 __signbit (x)                       :  \
                                 (sizeof (x) == DOUBLE_SIZE)         ?  \
                                 __signbitd (x)                      :  \
                                 __signbitf (x))
```


Data Types

```
enum NumberKind
{
   FP_SNAN = 0,                 /* signaling NaN */
   FP_QNAN,                     /* quiet NaN */
   FP_INFINITE,                 /* + or – infinity */
   FP_ZERO,                     /* + or – zero */
   FP_NORMAL,                   /* all normal numbers */
   FP_SUBNORMAL                 /* denormal numbers */
};

typedef short relop;

enum
{
   GREATERTHAN = ((relop) (0)),
   LESSTHAN,
   EQUALTO,
   UNORDERED
};

struct decimal
{
   char sgn;                        /* sign 0 for +, 1 for – */
   char unused;
   short exp;                       /* decimal exponent */
   struct
```

```
   {
      unsigned char length;
      unsigned char text[SIGDIGLEN];      /* significant digits */
      unsigned char unused;
   } sig;
};
typedef struct decimal decimal;

struct decform
{
   char style;                          /* FLOATDECIMAL or FIXEDDECIMAL */
   char unused;
   short digits;
};
typedef struct decform decform;

extern const double_t pi;
```

Functions

Trigonometric Functions

```
double_t cos                (double_t x);
double_t sin                (double_t x);
double_t tan                (double_t x);
double_t acos               (double_t x);  /* argument is in [0,pi] */
double_t asin               (double_t x);  /* argument is in [-pi/2,pi/2] */
double_t atan               (double_t x);  /* argument is in [-pi/2,pi/2] */

#ifdef powerc
long double cosl            (long double x);
long double sinl            (long double x);
long double tanl            (long double x);
long double acosl           (long double x); /*argument is in [0,pi]*/
long double asinl           (long double x); /*argument is in [-pi/2,pi/2]*/
long double atanl           (long double x); /*argument is in [-pi/2,pi/2]*/
#endif /* powerc */
```

```
double_t atan2              (double_t y, double_t x);

#ifdef powerc
long double atan2l          (long double y, long double x);
#endif /* powerc */
```

Hyperbolic Functions

```
double_t cosh               (double_t x);
double_t sinh               (double_t x);
double_t tanh               (double_t x);
double_t acosh              (double_t x);
double_t asinh              (double_t x);
double_t atanh              (double_t x);

#ifdef powerc
long double coshl           (long double x);
long double sinhl           (long double x);
long double tanhl           (long double x);
long double acoshl          (long double x);
long double asinhl          (long double x);
long double atanhl          (long double x);
#endif /* powerc */
```

Exponential Functions

```
double_t exp                (double_t x);

#ifdef powerc
long double expl            (long double x);
#endif /* powerc */

double_t expm1              (double_t x);

#ifdef powerc
long double expm1l          (long double x);
#endif /* powerc */
```

```
double_t exp2               (double_t x);
double_t frexp              (double_t x, int *exponent);
double_t ldexp              (double_t x, int n);
double_t log                (double_t x);

#ifdef powerc
long double exp2l           (long double x);
long double frexpl          (long double x, int *exponent);
long double ldexpl          (long double x, int n);
long double logl            (long double x);
#endif /* powerc */

double_t log2               (double_t x);

#ifdef powerc
long double log2l           (long double x);
#endif /* powerc */

double_t log1p              (double_t x);
double_t log10              (double_t x);

#ifdef powerc
long double log1pl          (long double x);
long double log10l          (long double x);
#endif /* powerc */

double_t logb               (double_t x);

#ifdef powerc
long double logbl           (long double x);
#endif /* powerc */

long double modfl           (long double x, long double *iptrl);
double modf                 (double x, double *iptr);
float modff                 (float x, float *iptrf);
```

```
double_t scalb              (double_t x, long int n);

#ifdef powerc
long double scalbl          (long double x, long int n);
#endif      /* powerc */
```

Power and Absolute Value Functions

```
double_t fabs               (double_t x);

#ifdef powerc
long double fabsl           (long double x);
#endif /* powerc */

double_t hypot              (double_t x, double_t y);
double_t pow                (double_t x, double_t y);
double_t sqrt               (double_t x);

#ifdef powerc
long double hypotl          (long double x, long double y);
long double powl            (long double x, long double y);
long double sqrtl           (long double x);
#endif /* powerc */
```

Gamma and Error Functions

```
double_t erf                (double_t x);           /* the error function */
double_t erfc               (double_t x); /* complementary error function */
double_t gamma              (double_t x);

#ifdef powerc
long double erfl            (long double x);        /* the error function */
long double erfcl           (long double x);/*complementary error function*/
long double gammal          (long double x);
#endif /* powerc */

double_t lgamma             (double_t x);

#ifdef powerc
long double lgammal         (long double x);
#endif /* powerc */
```

Nearest Integer Functions

```
double_t ceil               (double_t x);
double_t floor              (double_t x);

#ifdef powerc
long double ceill           (long double x);
long double floorl          (long double x);
#endif /* powerc */

double_t rint               (double_t x);

#ifdef powerc
long double rintl           (long double x);
#endif /* powerc */

double_t nearbyint          (double_t x);

#ifdef powerc
long double nearbyintl      (long double x);
#endif /* powerc */

long int rinttol            (double_t x);

#ifdef powerc
long int rinttoll           (long double x);
#endif /* powerc */

double_t round              (double_t x);

#ifdef powerc
long double roundl          (long double x);
#endif /* powerc */

long int roundtol           (double_t x);

#ifdef powerc
long int roundtoll          (long double x);
#endif /* powerc */
```

```
double_t trunc              (double_t x);

#ifdef powerc
long double truncl          (long double x);
#endif /* powerc */
```

Remainder Functions

```
double_t fmod               (double_t x, double_t y);
double_t remainder          (double_t x, double_t y);
double_t remquo             (double_t x, double_t y, int *quo);

#ifdef powerc
long double remainderl      (long double x, long double y);
long double remquol         (long double x, long double y, int *quo);
#endif /* powerc */
```

Auxiliary Functions

```
double_t copysign           (double_t x, double_t y);

#ifdef powerc
long double copysignl       (long double x, long double y);
#endif /* powerc */

long double nanl            (const char *tagp);
double nan                  (const char *tagp);
float nanf                  (const char *tagp);
long double nextafterl      (long double x, long double y);
double nextafterd           (double x, double y);
float nextafterf            (float x, float y);
```

Maximum, Minimum, and Positive Difference Functions

```
double_t fdim               (double_t x, double_t y);

#ifdef powerc
long double fdiml           (long double x, long double y);
#endif
```

```
double_t fmax               (double_t x, double_t y);
double_t fmin               (double_t x, double_t y);

#ifdef powerc
long double fmaxl           (long double x, long double y);
long double fminl           (long double x, long double y);
#endif
```

Internal Prototypes

```
long int __fpclassify       (long double x);
long int __fpclassifyd      (double x);
long int __fpclassifyf      (float x);
long int __isnormal         (long double x);
long int __isnormald        (double x);
long int __isnormalf        (float x);
long int __isfinite         (long double x);
long int __isfinited        (double x);
long int __isfinitef        (float x);
long int __isnan            (long double x);
long int __isnand           (double x);
long int __isnanf           (float x);
long int __signbit          (long double x);
long int __signbitd         (double x);
long int __signbitf         (float x);
double __inf                (void);
```

Non-NCEG Extensions

```
#ifndef  __NOEXTENSIONS__
```

Financial functions

```
double_t compound           (double_t rate, double_t periods);
double_t annuity            (double_t rate, double_t periods);
```

Random Function

```
double_t randomx            (double_t *x);
```

Relational Operator

```
relop relation              (double_t x, double_t y);

#ifdef powerc
relop relationl             (long double x, long double y);
#endif /* powerc */
```

Data Exchange Routines

```
#ifdef powerc
void x80told                (const extended80 *x80, long double *x);
void ldtox80                (const long double *x, extended80 *x80);
#endif /* powerc */
```

Binary-to-Decimal Conversions

```
void num2dec                (const decform *f, double_t x, decimal *d);

#ifdef powerc
void num2decl               (const decform *f, long double x, decimal *d);
#endif /* powerc */

double_t dec2num            (const decimal *d);
void dec2str                (const decform *f, const decimal *d, char *s);
void str2dec                (const char *s, short *ix, decimal *d,
                             short *vp);
#ifdef powerc
long double dec2numl        (const decimal *d);
#endif /* powerc */
float dec2f                 (const decimal *d);
short int dec2s             (const decimal *d);
long int dec2l              (const decimal *d);

#endif      /* __NOEXTENSIONS__ */
#endif
```

Floating-Point Environment Header File (fenv.h)

The file `fenv.h` defines a collection of functions designed to provide access to the floating-point environment for numerical programming. The file `fenv.h` declares many functions in support of numerical programming. It provides a set of environmental controls similar to the ones found in the SANE library.

Constants

```
#ifndef  __FENV__
#define  __FENV__
```

Floating-Point Exception Flags

```
#define  FE_INEXACT        0x02000000     /* inexact */
#define  FE_DIVBYZERO      0x04000000     /* divide-by-zero */
#define  FE_UNDERFLOW      0x08000000     /* underflow */
#define  FE_OVERFLOW       0x10000000     /* overflow */
#define  FE_INVALID        0x20000000     /* invalid */

/* The bitwise OR of all exception macros */

#define  FE_ALL_EXCEPT     (  FE_INEXACT | FE_DIVBYZERO | FE_UNDERFLOW | \
                              FE_OVERFLOW | FE_INVALID )
```

Rounding Direction Modes

```
#define  FE_TONEAREST      0x00000000
#define  FE_TOWARDZERO     0x00000001
#define  FE_UPWARD         0x00000002
#define  FE_DOWNWARD       0x00000003

#define  FE_DFL_ENV        &_FE_DFL_ENV   /* pointer to default environment*/
```

Data Types

```
typedef     long int    fenv_t;

typedef     long int    fexcept_t;

/* Definition of pointer to IEEE default environment object */

extern      fenv_t      _FE_DFL_ENV;         /* default environment object */
```

Functions

Controlling the Floating-Point Exceptions

```
void feclearexcept          (int excepts);
void fegetexcept            (fexcept_t *flagp, int excepts);
void feraiseexcept          (int excepts);
void fesetexcept            (const fexcept_t *flagp, int excepts);
int fetestexcept            (int excepts);
```

Controlling the Rounding Direction

```
int fegetround              (void);
int fesetround              (int round);
```

Controlling the Floating-Point Environment

```
void fegetenv               (fenv_t *envp);
int feholdexcept            (fenv_t *envp);
void fesetenv               (const fenv_t *envp);
void feupdateenv            (const fenv_t *envp);

#endif
```