NAME

feclearexcept, **fegetexceptflag**, **feraiseexcept**, **fesetexceptflag**, **fetestexcept**, **fegetround**, **fesetround**, **fegetenv**, **feholdexcept**, **fesetenv**, **feupdateenv**, **feenableexcept**, **fedisableexcept**, **fegetexcept** - floating-point environment control

LIBRARY

Math Library (libm, -lm)

SYNOPSIS

#include <fenv.h>

#pragma STDC FENV_ACCESS ON

int
feclearexcept(*int excepts*);

int
fegetexceptflag(*fexcept_t *flagp*, *int excepts*);

int
feraiseexcept(*int excepts*);

int
fesetexceptflag(*const fexcept_t *flagp*, *int excepts*);

int
fetestexcept(*int excepts*);

int
fegetround(*void*);

int
fesetround(*int round*);

int
fegetenv(*fenv_t *envp*);

int
feholdexcept(*fenv_t *envp*);

int
fesetenv(*const fenv_t *envp*);

int
feupdateenv(*const fenv_t *envp*);

int
feenableexcept(*int excepts*);

int
fedisableexcept(*int excepts*);

int
fegetexcept(*void*);

DESCRIPTION

The <*fenv.h*> routines manipulate the floating-point environment, which includes the exception flags and rounding modes defined in IEEE Std 754-1985.

Exceptions

Exception flags are set as side-effects of floating-point arithmetic operations and math library routines, and they remain set until explicitly cleared. The following macros expand to bit flags of type *int* representing the five standard floating-point exceptions.

FE_DIVBYZERO A divide-by-zero exception occurs when the *exact* result of a computation is infinite (according to the limit definition). For example, dividing a finite non-zero number by zero or computing **log**(*0*) raises a divide-by-zero exception.

FE_INEXACT An inexact exception is raised whenever there is a loss of accuracy due to rounding.

FE_INVALID Invalid operation exceptions occur when a program attempts to perform calculations for which there is no reasonable representable answer. For instance, subtraction of like-signed infinities, division of zero by zero, ordered comparison involving NaNs, and taking the real square root of a negative number are all invalid operations.

FE_OVERFLOW In contrast with divide-by-zero, an overflow exception occurs when an infinity is produced because the magnitude of the exact result is *finite* but too large to fit in the destination type. For example, computing DBL_MAX * 2 raises an overflow exception.

FE_UNDERFLOW
: Underflow occurs when the result of a computation loses precision because it is too close to zero. The result is a subnormal number or zero.

Additionally, the FE_ALL_EXCEPT macro expands to the bitwise OR of the above flags and any architecture-specific flags. Combinations of these flags are passed to the **feclearexcept**(), **fegetexceptflag**(), **feraiseexcept**(), **fesetexceptflag**(), and **fetestexcept**() functions to clear, save, raise, restore, and examine the processor's floating-point exception flags, respectively.

Exceptions may be *unmasked* with **feenableexcept**() and masked with **fedisableexcept**(). Unmasked exceptions cause a trap when they are produced, and all exceptions are masked by default. The current mask can be tested with **fegetexcept**().

Rounding Modes

IEEE Std 754-1985 specifies four rounding modes. These modes control the direction in which results are rounded from their exact values in order to fit them into binary floating-point variables. The four modes correspond with the following symbolic constants.

FE_TONEAREST
: Results are rounded to the closest representable value. If the exact result is exactly half way between two representable values, the value whose last binary digit is even (zero) is chosen. This is the default mode.

FE_DOWNWARD
: Results are rounded towards negative infinity.

FE_UPWARD
: Results are rounded towards positive infinity.

FE_TOWARDZERO
: Results are rounded towards zero.

The **fegetround**() and **fesetround**() functions query and set the rounding mode.

Environment Control

The **fegetenv**() and **fesetenv**() functions save and restore the floating-point environment, which includes exception flags, the current exception mask, the rounding mode, and possibly other implementation-specific state. The **feholdexcept**() function behaves like **fegetenv**(), but with the additional effect of clearing the exception flags and installing a *non-stop* mode. In non-stop mode, floating-point operations will set exception flags as usual, but no SIGFPE signals will be generated as a result. Non-stop mode is the default, but it may be altered by **feenableexcept**() and **fedisableexcept**(). The **feupdateenv**() function restores a saved environment similarly to **fesetenv**(), but it also re-raises any floating-point exceptions from the old environment.

The macro FE_DFL_ENV expands to a pointer to the default environment.

EXAMPLES

The following routine computes the square root function. It explicitly raises an invalid exception on appropriate inputs using **feraiseexcept**(). It also defers inexact exceptions while it computes intermediate values, and then it allows an inexact exception to be raised only if the final answer is inexact.

```
#pragma STDC FENV_ACCESS ON
double sqrt(double n) {
        double x = 1.0;
        fenv_t env;

        if (isnan(n) || n < 0.0) {
                feraiseexcept(FE_INVALID);
                return (NAN);
        }
        if (isinf(n) || n == 0.0)
                return (n);
        feholdexcept(&env);
        while (fabs((x * x) - n) > DBL_EPSILON * 2 * x)
                x = (x / 2) + (n / (2 * x));
        if (x * x == n)
                feclearexcept(FE_INEXACT);
        feupdateenv(&env);
        return (x);
}
```

SEE ALSO

cc(1), feclearexcept(3), fedisableexcept(3), feenableexcept(3), fegetenv(3), fegetexcept(3), fegetexceptflag(3), fegetround(3), feholdexcept(3), feraiseexcept(3), fesetenv(3), fesetexceptflag(3), fesetround(3), fetestexcept(3), feupdateenv(3), fpgetprec(3), fpsetprec(3)

STANDARDS

Except as noted below, <*fenv.h*> conforms to ISO/IEC 9899:1999 ("ISO C99"). The **feenableexcept**(), **fedisableexcept**(), and **fegetexcept**() routines are extensions.

HISTORY

The <*fenv.h*> header first appeared in FreeBSD 5.3. It supersedes the non-standard routines defined in <*ieeefp.h*> and documented in fpgetround(3).

CAVEATS

The FENV_ACCESS pragma can be enabled with

```
#pragma STDC FENV_ACCESS ON
```

and disabled with the

```
#pragma STDC FENV_ACCESS OFF
```

directive. This lexically-scoped annotation tells the compiler that the program may access the floating-point environment, so optimizations that would violate strict IEEE-754 semantics are disabled. If execution reaches a block of code for which FENV_ACCESS is off, the floating-point environment will become undefined.

BUGS

The FENV_ACCESS pragma is unimplemented in the system compiler. However, non-constant expressions generally produce the correct side-effects at low optimization levels.