fabs/fabsf

compute floating point absolute value

SYNOPSIS

```
#include <math.h>

double fabs(double x)

float fabsf(float x)
```

DESCRIPTION

The **fabs** function returns the floating point absolute value of a floating point argument **x**. The **fabsf** function does the same thing for a single precision argument.

On floating point Transputers this function is implemented "inline". To get a functional version, precede your call with:

```
#undef fabs
```

Or,

```
#undef fabsf
```

RETURN VALUE

See above

RELATED FUNCTIONS

abs, labs

fclose/fcloseall

close streams

SYNOPSIS

```
#include <stdio.h>

int fclose(FILE *stream)

int fcloseall(void)
```

DESCRIPTION

The **fclose** function causes any buffered data for the specified **stream** to be written out, and the **stream** to be closed. The **fcloseall** function does the same thing for all currently open streams.

Note that termination of a program via "exit", or a return from "main", also closes all open files.

RETURN VALUE

These functions return EOF to indicate an error. The **fclose** function returns zero for success. The **fcloseall** function returns the number of open streams which were closed.

RELATED FUNCTIONS

close, fflush, fopen

fdopen

convert handle to stream

SYNOPSIS

```
#include <stdio.h>

FILE *fdopen(int handle, const char *mode)
```

DESCRIPTION

The **fdopen** function associates a stream with a file previously opened with **handle**. The allowed format for the **mode** string is the same as used with the "fopen" function, but must not conflict with the original access permission when the file was opened using "open" or "creat".

As this function is not supported in the ANSI standard library, its use in new work is discouraged (see the "fopen", "fclose", "fread", and "fwrite" functions for the recommended replacements).

RETURN VALUE

The **fdopen** function returns NULL on error, otherwise a pointer to the opened stream.

RELATED FUNCTIONS

fileno, fopen

feof/ferror

check stream status

SYNOPSIS

```
#include <stdio.h>

int feof(FILE *stream)

int ferror(FILE *stream)
```

DESCRIPTION

The **feof** function returns a non-zero value if EOF has been reached for the specified **stream** (assumed to be open for reading). The **ferror** function returns non-zero when an error has been detected during reading or writing of the specified **stream**.

RETURN VALUE

See above. The functions return non-zero if the predicate being tested is TRUE, zero if FALSE.

RELATED FUNCTIONS

clearerr

fflush

flush stream

SYNOPSIS

#include <stdio.h>

int fflush(FILE *stream)

DESCRIPTION

The **fflush** function causes any buffered data for the named stream to be written, if the stream is open for writing; or purged, if the stream is open for reading. The **fflush** call also purges anything buffered by a prior "ungetc" call.

RETURN VALUE

The fflush function returns EOF on error and zero for success.

RELATED FUNCTIONS

fclose

fft/fftf

compute forward FFT transform

SYNOPSIS

#include <math.h>

void fft(COMPLEX x[], int logsize)

void fftf(COMPLEXF x[], int logsize)

DESCRIPTION

The **fft** function performs a radix-2, forward, Fast Fourier Transform on an input array of complex floating point numbers (**x**). The **fftf** function does the same thing for an array of single precision floating point numbers. The transform is done in-place, so the output data is also stored into **x**. The **logsize** parameter is the log (base 2), of the number of complex numbers in the array. In the following definition, N is equal to the number of elements in the array (2**__logsize__):

$$x'(k) = \sum_{n=0}^{N-1} x(n) e^{-j(2(PI)/N)kn}$$

Note that there is considerable disagreement in the literature about what constitutes the "forward" transform, and what should be called the "inverse" (see the "ifft" routine description). Also, for a complete forward-inverse transform to produce the original data values, a scaling factor of 1/N must be applied to the data. Some authorities apply it all on the forward or inverse transforms, others apply 1/sqrt(N) to each. With this in mind, the **fft** and "ifft" functions do not perform any scaling. If your application requires it, you must apply it directly to the individual data points produced by one of the transforms.

RETURN VALUE

None. Note that for floating point Transputers, the **fftf** function has been hand optimized, for speed, in assembly language.

RELATED FUNCTIONS

ifft

fgetc

read character

SYNOPSIS

```
#include <stdio.h>

int fgetc(FILE *stream)
```

DESCRIPTION

The **fgetc** function obtains the next character (if available), from the input **stream**. The "getc" macro is functionally equivalent (ignoring side effects).

RETURN VALUE

The **fgetc** function returns EOF on error, or the character read. The character is treated as unsigned for purposes of distinguishing it from the error return.

RELATED FUNCTIONS

fputc, getc, getchar, putc, putchar, ungetc

fgets

read a line

SYNOPSIS

```
#include <stdio.h>

char *fgets(char *ptr, int n, FILE *stream)
```

DESCRIPTION

The **fgets** function reads at most **n**-1 characters from the **stream**, and places them in memory starting at **ptr**. A newline, or EOF, also terminates the read operation (the newline is retained). A '\0' character is added after the last character read.

RETURN VALUE

The **fgets** function returns NULL if an error was detected (or EOF found before any characters were read). On success **ptr** is returned.

RELATED FUNCTIONS

fputs, gets, puts

fileno

convert stream to handle

SYNOPSIS

```
#include <stdio.h>

int fileno(FILE *stream)
```

DESCRIPTION

The **fileno** function converts a **stream** into the corresponding file handle.

As this function is not supported in the ANSI standard library, its use in new work is discouraged (see the "fopen", "fclose", "fread", and "fwrite" functions for the recommended replacements).

RETURN VALUE

See above

RELATED FUNCTIONS

fdopen

floor/floorf

compute "floor"

SYNOPSIS

#include <math.h>

double floor(double x)

float floorf(float x)

DESCRIPTION

The **floor** and **floorf** functions return the largest floating point integral value not greater than **x**.

RETURN VALUE

See above

RELATED FUNCTIONS

ceil, ceilf

fmod/fmodf

compute floating point remainder

SYNOPSIS

#include <math.h>

double fmod(double x, double y)

float fmodf(float x, float y)

DESCRIPTION

The **fmod** and **fmodf** functions compute the floating point remainder of **x/y**.

RETURN VALUE

The functions return the value **x** - (i * **y**), for some integer i such that if **y** is non-zero, the result has the same sign as **x** and magnitude less than the magnitude of **y**. If **y** is zero the results are undefined.

RELATED FUNCTIONS

None

fopen

open stream

SYNOPSIS

#include <stdio.h>

FILE *fopen(const char *path, const char *mode)

DESCRIPTION

The **fopen** function opens the file specified by **path**, for reading and/or writing, as indicated by **mode**. The mode string specifies the allowed operations (all settings assume the file will be text and CR/LF mapping may be used by the host OS if necessary):

"a" - Open the file for appending (at the end). The file is created if it doesn't exist.
"a+" - Open the file for reading and appending (at the end). The file is created if it doesn't exist.
"r" - Open the file for reading (the file must already exist).
"r+" - Open the file for reading and writing (the file must already exist).
"w" - Open/create the file for writing. If the file exists the contents are purged.
"w+" - Open/create the file for reading and writing. If the file exists the contents are purged.

To open a "binary" file without any CR/LF mapping nonsense, append a 'b' character at the end of the mode string (or optionally before the '+', if one is used). For example, to open a binary file in "r+" mode, either of the following are legal:

"r+b" or "rb+"

RETURN VALUE

The **fopen** function returns NULL on error, otherwise the new stream pointer.

RELATED FUNCTIONS

fclose

fprintf

formatted write to stream

SYNOPSIS

```
#include <stdio.h>

int fprintf(FILE *stream, const char *format, ...)
```

DESCRIPTION

The **fprintf** function writes to the specified **stream**. The write is formatted under the control of the **format** argument, which specifies how subsequent arguments (if any), are to be converted for output. See the "printf" function description for more information about formatting options.

RETURN VALUE

The number of characters written (negative values indicate an error).

RELATED FUNCTIONS

printf, sprintf, vfprintf, vprintf, vsprintf

fputc

write a character

SYNOPSIS

#include <stdio.h>

int fputc(int c, FILE *stream)

DESCRIPTION

The **fputc** function writes the character specified by c to the output **stream**.

RETURN VALUE

The **fputc** function returns the character written (or EOF on error).

RELATED FUNCTIONS

fgetc, getc, getchar, putc, putchar, ungetc

fputs

write a line

SYNOPSIS

```
#include <stdio.h>

int fputs(const char *ptr, FILE *stream)
```

DESCRIPTION

The **fputs** function writes a '\0' terminated string (pointed to by **ptr**), to the specified **stream**. The '\0' is not written.

RETURN VALUE

The **fputs** function returns EOF if an error occurs; otherwise a non-negative value.

RELATED FUNCTIONS

fgets, gets, puts

frand/frandf

floating point random number

SYNOPSIS

#include <stdlib.h>

double frand(void)

float frandf(void)

DESCRIPTION

The **frand** function returns a sequence of pseudo-random floating point values in the range of zero to one (inclusive). The **frand** function uses the same seed as the integer "rand" function, and thus, may also be initialized using the "srand" function. The **frandf** function does the same thing as **frand**, but produces a single precision result.

RETURN VALUE

See above

RELATED FUNCTIONS

rand, srand

fread

read from stream

SYNOPSIS

#include <stdio.h>

size_t fread(void *ptr, size_t size, size_t nmembs, FILE *stream)

DESCRIPTION

The **fread** function reads **nmembs** worth of data items, each of **size** bytes, from **stream**, into memory starting at the address specified by **ptr**.

RETURN VALUE

The number of data items successfully read. This number may be less than **nmembs** if an error or EOF is encountered.

RELATED FUNCTIONS

fwrite

free

free heap memory

SYNOPSIS

```
#include <stdlib.h>

void free(void *ptr)
```

DESCRIPTION

The **free** function returns the previously allocated region of heap memory pointed to by **ptr** to the heap free storage pool.

The value of **ptr** must be the result of an earlier call to "calloc", "malloc", or "realloc"; or the results are undefined.

The **free** function is equivalent to the "cfree" function.

RETURN VALUE

None

RELATED FUNCTIONS

addfree, calloc, cfree, malloc, realloc

freopen

redirect stream

SYNOPSIS

#include <stdio.h>

FILE *freopen(const char *path, const char *mode, FILE *stream)

DESCRIPTION

The **freopen** function closes whatever is associated with **stream** and opens the file specified by **path** and assigns **stream** to it. The allowed format for the **mode** string is the same as used with the "fopen" function.

This function is conventionally used to redirect the streams which are automatically opened for the user program ("stdio", "stdout", and "stderr").

RETURN VALUE

The **freopen** function returns NULL on error and closes the original stream. On success, **freopen** returns a pointer to the "new" stream.

RELATED FUNCTIONS

fopen

frexp/frexpf

decompose floating point number

SYNOPSIS

#include <math.h>

double frexp(double value, int *exp)

float frexpf(float value, int *exp)

DESCRIPTION

The **frexp** function breaks a floating point **value** into a normalized fraction and a integral power of two. The integral value is stored in the "int" pointed to by **exp**. The **frexpf** function does the same thing for a single precision argument.

RETURN VALUE

These functions return a floating point number, such that the number has magnitude in the range [1/2,1] or zero, and **value** equals the number raised to the power ***exp**. If **value** is zero, both parts of the result are also.

RELATED FUNCTIONS

ldexp

fscanf

formatted read from stream

SYNOPSIS

```
#include <stdio.h>

int fscanf(FILE *stream, const char *format, ...)
```

DESCRIPTION

The **fscanf** function reads from the specified **stream**. The read is formatted under the control of the **format** argument, which specifies the legal input text sequences and conversion instructions. Subsequent arguments (if any), are used as pointers to the objects which receive converted data from the input. See the "scanf" function description for more information about formatting options.

RETURN VALUE

The **fscanf** function returns EOF if an input failure occurs before any conversions. Otherwise, **fscanf** returns the number of input sequences which were matched, converted, and assigned. The number of sequences returned, may be less than the number provided for in the **format** string, if a match fails, or EOF is encountered.

If EOF occurs in the middle of matching an input sequence, such that the match may already be considered successful (not counting optionally matched whitespace), the EOF is considered to merely terminate the current match. Any remaining conversions, however, will be immediately aborted if they require input.

RELATED FUNCTIONS

scanf, sscanf, strtod, strtol, strtoul

fseek

change position within stream

SYNOPSIS

```
#include <stdio.h>

int fseek(FILE *stream, long int offset, int origin)
```

DESCRIPTION

The **fseek** function changes the current read/write position within the specified **stream**. The new position is computed as an **offset** (in bytes), from the measuring position specified by **origin**. Three macro definitions in "stdio.h" indicate the legal values for **origin**:

1. SEEK_SET - The value of **offset** is measured from the start of the stream.

2. SEEK_CUR - The value of **offset** is measured from the current position in the stream.

3. SEEK_END - The value of **offset** is measured from the end of the stream.

Note that for host environments which perform CR/LF mapping, seeks on "text" streams may produces unexpected results unless the value used for **offset** is either zero, or was returned from a previous call to the "ftell" function. If a value returned from a "ftell" call is used, the **origin** value to the **fseek** call should be "SEEK_CUR".

RETURN VALUE

Zero on success, non-zero if the request is illegal. An example of an illegal request: calling **fseek** with an **origin** value of "SEEK_SET", and a negative **offset**, (seeking before the beginning of a file).

RELATED FUNCTIONS

ftell, lseek, rewind

ftell

report current position within stream

SYNOPSIS

```
#include <stdio.h>

long int ftell(FILE *stream)
```

DESCRIPTION

The **ftell** function reports the current read/write position within the specified **stream**. For a binary stream, this value is the number of bytes from the beginning of the file. If the host environment performs CR/LF mapping within "text" streams, the reported value has no simple definition, and is mainly used to keep track of a particular stream position which will later be returned to using a "fseek" function call.

RETURN VALUE

The current stream position value is returned on success, (-1L) on failure.

RELATED FUNCTIONS

fseek, lseek, rewind

fwrite

write to stream

SYNOPSIS

#include <stdio.h>

size_t fwrite(const void *ptr, size_t size, size_t nmembs, FILE *stream)

DESCRIPTION

The **fwrite** function writes **nmembs** worth of data items, each of **size** bytes, to **stream**, from memory starting at the address specified by **ptr**.

RETURN VALUE

The number of data items successfully written. This number may be less than **nmembs** if an error is encountered.

RELATED FUNCTIONS

fread

getc/getch/getchar/getche

read a character

SYNOPSIS

```
#include <stdio.h>

int getc(FILE *stream)

int getch(void)

int getchar(void)

int getche(void)
```

DESCRIPTION

The **getc** macro obtains the next character (if available), from the input **stream**. Since **getc** is defined to be a macro, the stream argument may be evaluated more than once, and must not have side-effects. To get a functional version, use either the "fgetc" equivalent, or precede the function call with:

```
#undef getc
```

The **getch** function is probably specific to MS-DOS. It reads, without echoing, a single character from the console. When reading a cursor key, or function key, **getch** must be called twice (the second call gets the actual scan code).

The **getchar** macro obtains the next character (if available), from the input stream "stdin". To get a functional version, use either the "fgetc(stdin)" equivalent, or precede the function call with:

```
#undef getchar
```

The **getche** function is probably specific to MS-DOS. It reads, and echoes, a single character from the console. When reading a cursor key, or function key, **getche** must be called twice (the second call gets the actual scan code).

RETURN VALUE

The **getch** and **getche** functions returns the character, or portion thereof, read. There is no error return.

The **getc** and **getchar** macros/functions return EOF on error, or the character read. The characters are treated as unsigned for purposes of distinguishing them from the error return.

RELATED FUNCTIONS

fgetc, fputc, putc, putchar, ungetc

getenv

read environment value

SYNOPSIS

```
#include <stdlib.h>

char *getenv(const char *name)
```

DESCRIPTION

The **getenv** function searches an "environment list", provided by the host environment, for a string that matches **name**. If a matching string is found, a pointer is returned to the string "value" associated with it. The memory associated with the returned value should not be modified by the programmer. Since it is possible for the host environment to change, subsequent calls to **getenv** (for the same **name**), may return different pointers (and assigned values).

In the **Transputer Toolset** implementation of **getenv**, the "value" is stored in memory obtained from the heap using the "malloc" function. If necessary, the memory may be returned to the heap using a call to "free" (with the pointer returned from the original call to **getenv**).

RETURN VALUE

See above. If no match for **name** is found, **getenv** returns NULL.

RELATED FUNCTIONS

None

GetHiPriQ/GetLoPriQ

save queue pointers

SYNOPSIS

```
#include <conc.h>

void GetHiPriQ(int ptrs[])

void GetLoPriQ(int ptrs[])
```

DESCRIPTION

These functions provide "C" bindings for the Transputer "saveh" (**GetHiPriQ**), and "savel" (**GetLoPriQ**), instructions. The **ptrs** array will have the queue "front" pointer stored in **ptrs**[0], and the queue "back" pointer in **ptrs**[1] upon return. See the appropriate INMOS documentation for more information about the "saveX" instructions.

These functions are implemented "inline". To get a functional version, precede your call with:

```
#undef GetHiPriQ
```

Or,

```
#undef GetLoPriQ
```

RETURN VALUE

None

RELATED FUNCTIONS

SetHiPriQ, SetLoPriQ

gets

read a line

SYNOPSIS

```
#include <stdio.h>

char *gets(char *ptr)
```

DESCRIPTION

The **gets** function reads characters from the input stream "stdin", into memory starting at **ptr**. Characters are read until EOF, or a newline character is read (it is discarded). A '\0' character is added after the last character read.

You must ensure that the character array (pointed to by **ptr**), is large enough to hold the worst case input line, otherwise corruption of other data or code is likely. For new work we suggest you use the "fgets" function instead.

RETURN VALUE

The **gets** function returns NULL if an error was detected (or EOF was found before any characters were read). On success **ptr** is returned.

RELATED FUNCTIONS

fgets, fputs, puts

gmtime

determine Coordinated Universal Time

SYNOPSIS

```
#include <time.h>

struct tm *gmtime(const time_t *timer)
```

DESCRIPTION

The **gmtime** function converts the calendar time specified by **timer** into UTC time, and stores the resulting broken-down time into an internal static data area.

RETURN VALUE

The **gmtime** function returns a pointer to the converted broken-down time. The broken-down time data area is static, and is shared with the "asctime", "ctime" and "localtime" functions. The data will be overwritten upon subsequent calls to any of these four functions.

RELATED FUNCTIONS

asctime, ctime, localtime, time

hypot/hypotf

compute Euclidean distance

SYNOPSIS

```
#include <math.h>

double hypot(double x, double y)

float hypotf(float x, float y)
```

DESCRIPTION

The **hypot** function returns the length, of the hypotenuse, of a right triangle whose sides have lengths **x** and **y**. The **hypotf** function does the same thing for single precision arguments. A call to these functions is roughly equivalent to a "sqrt" function call with the following arguments:

```
sqrt(x*x + y*y)
```

Note that the equivalent "sqrt" function call will often be less accurate than a call to **hypot**; particularly when the magnitudes of **x** and **y** are greatly different.

RETURN VALUE

See above. A range error ("errno" is set to ERANGE), or domain error ("errno" is set to EDOM), may occur depending on the parameter values.

RELATED FUNCTIONS

sqrt

HSemP/_HSemP/HSemV/_HSemV

mixed-priority semaphores

SYNOPSIS

```
#include <conc.h>

int HSemP(Semaphore s)

int _HSemP(Semaphore *s)

int HSemV(Semaphore s)

int _HSemV(Semaphore *s)
```

DESCRIPTION

The **HSemP** and **HSemV** macros implement semaphore operations on the Transputer. They are equivalent to the "SemP" and "SemV" functions, but may be used where processes at mixed priority levels will be sharing the same semaphore (**s**). These functions ensure correct mixed priority operation by internally switching to high priority during the semaphore access (with a somewhat higher overhead than the mono-priority versions). The **_HSemV** and **_HSemP** functions are primitives used by the macros, and are generally not called directly.

See the description of the "SemP" and "SemV" functions for more information about using these macros.

RETURN VALUE

These functions return the "use" field from the **s** "Semaphore" structure.

RELATED FUNCTIONS

SemAlloc, SemFree, SemP, SemV

ifft/ifftf

compute inverse FFT transform

SYNOPSIS

#include <math.h>

void ifft(COMPLEX x[], int logsize)

void ifftf(COMPLEXF x[], int logsize)

DESCRIPTION

The **ifft** function performs a radix-2, inverse, Fast Fourier Transform on an input array of complex floating point numbers (**x**). The **ifftf** function does the same thing for an array of single precision floating point numbers. The transform is done in-place, so the output data is also stored into **x**. The **logsize** parameter is the log (base 2), of the number of complex numbers in the array. In the following definition, N is equal to the number of elements in the array (2**__logsize__):

$$x'(k) = \sum_{n=0}^{N-1} x(n) e^{j(2(PI)/N)kn}$$

Note that there is considerable disagreement in the literature about what constitutes the "forward" transform, and what should be called the "inverse" (see the "fft" routine description). Also, for a complete forward-inverse transform pair to produce the original data values, a scaling factor of 1/N must be applied to the data. Some authorities apply it all on the forward or inverse transforms, others apply 1/sqrt(N) to each. With this in mind, the **ifft** and "fft" functions do not perform any scaling. If your application requires it, you must apply it directly to the individual data points produced by one of the transforms.

RETURN VALUE

None. Note that for floating point Transputers, the **ifftf** function has been hand optimized, for speed, in assembly language.

RELATED FUNCTIONS

fft

index

search string for character

SYNOPSIS

#include <string.h>

char *index(const char *ptr, int c)

DESCRIPTION

The **index** function searches a string, pointed to by **ptr**, for the first occurrence of the character **c**. A pointer to where the character was found is returned, or NULL, if a '\0' string terminator was found instead. The '\0' character is considered part of the string and may be searched for. This function is called "strchr" in ANSI libraries, and use of that name is recommended for all new work.

RETURN VALUE

See above

RELATED FUNCTIONS

rindex, strchr, strrchr

is*

classify characters

isalnum
isalpha
isascii
iscntrl
isdigit
isgraph
islower
isprint
ispunct
isspace
isupper
isxdigit

SYNOPSIS

```
#include <ctype.h>

int isalnum(int c)

int isalpha(int c)

int isascii(int c)

int iscntrl(int c)

int isdigit(int c)

int isgraph(int c)

int islower(int c)

int isprint(int c)

int ispunct(int c)

int isspace(int c)

int isupper(int c)

int isxdigit(int c)
```

DESCRIPTION

These macros classify an argument, **c**, as to whether it belongs in a particular set. The macros are designed to evaluate the argument only once, and are safe for use with arguments with side-effects.

The following descriptions cover the individual macros (and indicate what combination of the other macros is equivalent):

isalnum - True for alphanumeric characters (**isalpha || isdigit**).

isalpha - True for lower or upper case letters (**islower || isupper**).

isascii - True for ASCII characters (0x00-0x7F).

iscntrl - True for control characters (0x00-0x1F, 0x7F).

isdigit - True for a decimal digit ('0'-'9').

isgraph - True for printing characters except space (0x21-0x7E).

islower - True for lower case letters ('a'-'z').

isprint - True for printing characters (0x20-0x7E).

ispunct - True for printing characters other than space (and except anything **isalnum** is true for).

isspace - True for whitespace characters (' ', '\f', '\n', '\r', '\t', '\v').

isupper - True for upper case letters ('A'-'Z').

isxdigit - True for hexadecimal digits ('0'-'9', 'a'-'f', 'A'-'F').

RETURN VALUE

All the macros evaluate to a non-zero truth value if the predicate is satisfied; otherwise zero to indicate failure.

RELATED FUNCTIONS

toascii, tolower, toupper

isort

insertion sort

SYNOPSIS

#include <stdlib.h>

void isort(void *base, size_t nmemb, size_t size, int (*compare)(const void *, const void *))

DESCRIPTION

The **isort** function is an implementation of the "insertion sort" algorithm which has the same calling sequence as the standard "qsort" function. This function may be used to advantage (compared to "qsort"), when the number of items to be sorted is small (less than 10 or 20).

See the "qsort" function description for detailed parameter explanations.

RETURN VALUE

None

RELATED FUNCTIONS

qsort, ssort

kbhit

check console for keystroke

SYNOPSIS

```
#include <stdio.h>

int kbhit(void)
```

DESCRIPTION

The **kbhit** function returns a non-zero value if a key has been pressed on the system console, zero if not. This function is not part of the ANSI library, and isn't supported under many host environments.

RETURN VALUE

See above

RELATED FUNCTIONS

None

labs

compute long absolute value

SYNOPSIS

#include <stdlib.h>

long labs(long l)

DESCRIPTION

The **labs** function returns the long integer absolute value of a long integer argument **l**.

RETURN VALUE

See above

RELATED FUNCTIONS

abs, fabs, fabsf

ldexp/ldexpf

compose floating point number

SYNOPSIS

```
#include <math.h>

double ldexp(double value, int exp)

float ldexpf(float value, int exp)
```

DESCRIPTION

The **ldexp** function builds a floating point number by multiplying the **value** by the quantity 2****exp**. The **ldexpf** function does the same thing for a single precision argument.

RETURN VALUE

See above. A range error ("errno" is set to ERANGE), occurs if the number is not representable in the floating point format.

RELATED FUNCTIONS

frexp

ldiv

compute long integer quotient and remainder

SYNOPSIS

```
#include <stdlib.h>

ldiv_t ldiv(long int num, long int denom)
```

DESCRIPTION

The **ldiv** function returns the long integer quotient and remainder of dividing **num** by **denom**. If the division is inexact, the sign of the quotient will be the same as the algebraic quotient, and the magnitude of the quotient will be the largest integer less than the magnitude of the algebraic quotient. If the result can be represented, then:

num = quotient * **denom** + remainder

RETURN VALUE

The quotient and remainder are returned in a structure of type "ldiv_t", which has two long integer members, "quot" and "rem".

RELATED FUNCTIONS

div

localtime

determine local time

SYNOPSIS

```
#include <time.h>

struct tm *localtime(const time_t *timer)
```

DESCRIPTION

The **localtime** function converts the calendar time specified by **timer**, into local time, and stores the resulting broken-down time into an internal static data area.

RETURN VALUE

The **localtime** function returns a pointer to the converted broken-down time. The broken-down time data area is static, and is shared with the "asctime", "ctime" and "gmtime" functions. The data will be overwritten upon subsequent calls to any of these four functions.

RELATED FUNCTIONS

asctime, ctime, gmtime, time

log/logf

compute natural logarithm

SYNOPSIS

```
#include <math.h>

double log(double x)

float logf(float x)
```

DESCRIPTION

The **log** function returns the floating point natural logarithm of **x**. The **logf** function does the same thing for a single precision argument.

RETURN VALUE

A domain error ("errno" set to EDOM), occurs if **x** is negative. A range error ("errno" is set to ERANGE), occurs if **x** is zero.

RELATED FUNCTIONS

exp, log10, pow

log10/log10f

compute base-10 logarithm

SYNOPSIS

```
#include <math.h>

double log10(double x)

float log10f(float x)
```

DESCRIPTION

The **log10** function returns the floating point base-10 logarithm of **x**. The **log10f** function does the same thing for a single precision argument.

RETURN VALUE

A domain error ("errno" set to EDOM), occurs if **x** is negative. A range error ("errno" is set to ERANGE), occurs if **x** is zero.

RELATED FUNCTIONS

exp, log, pow

longjmp

non-local jump

SYNOPSIS

```
#include <setjmp.h>

void longjmp(jmp_buf env, int val)
```

DESCRIPTION

A call to the **longjmp** function causes the current function and stack environment to be replaced with the conditions which existed when the most recent call to the "setjmp" function occurred which used the same "jmp_buf" structure (**env**). If the "setjmp" function hasn't been called with **env**, or was not called from a function earlier in the set of nested calls, the results are undefined.

After the **longjmp** call is completed, all local variable information present at the previous call to "setjmp" is available, although changes which occurred chronologically after the "setjmp" call will generally be in effect.

RETURN VALUE

After the **longjmp** call, the program continues execution as if it had just returned from "setjmp" with a return value of **val**. Since the return value of "setjmp" is used to distinguish between the initial registration call to "setjmp", and the subsequent activation through the **longjmp** call, the value of **val** must not be zero (it will be mapped into a value of one automatically if zero is used).

RELATED FUNCTIONS

setjmp

lseek

change position within file

SYNOPSIS

#include <stdio.h>

long int lseek(int handle, long int offset, int origin)

DESCRIPTION

The **lseek** function changes the current read/write position within the file specified by **handle**. The new position is computed as an **offset** (in bytes), from the measuring position specified by **origin**. Three macro definitions in "stdio.h" indicate the legal values for **origin**:

1. SEEK_SET - The value of **offset** is measured from the start of the file.

2. SEEK_CUR - The value of **offset** is measured from the current position in the file.

3. SEEK_END - The value of **offset** is measured from the end of the file.

The **lseek** function is not supported in the ANSI standard library, thus use in new work is discouraged (the "fseek" function is the recommended replacement).

RETURN VALUE

The new read/write position (in bytes), as measured from the start of the file. A negative return value indicates an error.

RELATED FUNCTIONS

close, creat, fseek, ftell, open, read, rewind, write