_country

ROM based country identifier

Class: GEMDOS

Category: Process Environment

SYNOPSIS

```
    #include <dos.h>

    extern enum {} _country;  country identifier
```

DESCRIPTION

These variable gives the country for which the operating system is nationalised. The currently used values are:

Value	Identifier	Country
0	USA	USA
1	FRG	Germany
2	FRA	France
3	GBR	Great Britain
4	SPA	Spain
5	ITA	Italy
6	SWE	Sweden
7	SWF	Switzerland (French)
8	SWG	Switzerland (German)
9	TUR	Turkey
10	FIN	Finland
11	NOR	Norway
12	DEN	Denmark
13	SAU	Saudi Arabia
14	HOL	Holland

cprintf

Formatted print to console

Class: Lattice

Category: Formatted I/O

SYNOPSIS

```
    #include <conio.h>

    length = cprintf(fmt,arg1,arg2,...);

    int length;         number of characters generated
    const char *fmt;    format string

    See printf for arg1, arg2, and so on.
```

DESCRIPTION

The printf group of functions generate a stream of ASCII characters by analysing the format string and performing various conversion operations on the remaining arguments. The cprintf form of printf sends the stream to the console via a low-level operating system interface, thereby eliminating the buffered I/O overhead.

See the description of the printf function for a complete discussion of the arguments and conversion specifications.

RETURNS

This function returns the number of output characters generated.

SEE

fprintf, lprintf, printf, sprintf, vfprintf, vprintf, vsprintf

cputc, cputs

Console output operations

Class: Lattice *Category: Console and Port I/O*

SYNOPSIS

```
#include <dos.h>

c = cputc(c);               put character to console
count = cputs(buffer);      put string to console

int c;                      input character
int count;                  output character count
const char *buffer;         pointer to input string
```

DESCRIPTION

These functions put single characters or character strings to the console display. They are similar to putchar and puts except that they call the low-level video routines instead of working through the File Manager. This can result in better display performance.

RETURNS

The cputc function returns the character that was used as its argument, while cputs returns the number of characters sent to the display.

SEE

cprintf, putchar, puts, kbhit

creat

Create a file

Class: UNIX *Category: Low-Level I/O*

SYNOPSIS

```
#include <fcntl.h>

fh = creat(name,prot);

int fh;                file handle
const char *name;      file name
int prot;              protection mode
```

DESCRIPTION

This function is exactly the same as calling the open function in the following way:

```
open(name,O_WRONLY | O_TRUNC | O_CREAT |
  (prot & O_RAW),(prot & ~O_RAW));
```

In other words, the file is created if it doesn't exist and truncated if it does exist. Then it is opened for writing, and the translation mode is picked up from the prot argument. The protection mode can be any of the following:

Value	Meaning
S_IWRITE	Write permission
S_IREAD	Read permission
S_IWRITE I S_IREAD	Write and read permission

Also you can OR in O_RAW to suppress file translation. For instance, if prot is

```
O_RAW | S_IREAD
```

the file will be created as read-only and will be processed in raw (untranslated) mode. The read-only condition takes effect only if a new file must be created; if the file already exists, its protection mode is unchanged. Also, you can write to a newly-created read-only file until you close it for the first time.

RETURNS

If the operation succeeds, a file handle is returned, which is a positive integer. Otherwise it returns -1 and places error information in errno and _OSERR.

SEE

Fcreate, errno, _OSERR, chgfa, chmod, close, open

cscanf

Formatted input from console

Class: Lattice

Category: Formatted I/O

SYNOPSIS

```
#include <stdio.h>

n = cscanf(fmt,arg1,arg2,...);

int n;              number of input items matched, or
                    EOF
const char *fmt;    format string
void *argx;         pointers to input data areas
                    (x=1,2...)
```

DESCRIPTION

The cscanf function performs formatted input conversions on text obtained from the system console. The input characters are read and checked against the format string. The description of the scanf function fully describes the formats and conversion specifications.

RETURNS

The function returns the number of assignments that were made. For example, a return value of 3 indicates that conversion results were assigned to arg1, arg2, and arg3.

SEE

fscanf, scanf, sscanf

ctime

Convert time value to string

Class: ANSI

Category: Date and Time

SYNOPSIS

```
#include <time.h>

s = ctime(t);

char *s;            points to time string
const time_t *t;    points to time value
```

DESCRIPTION

This function converts a Greenwich Mean Time (GMT) time value to an ASCII string of *exactly* 26 characters having the form:

```
DDD MMM dd hh:mm:ss YYYY\n\0
```

where DDD is the day of the week, MMM is the month, dd is the day of the month, hh:mm:ss is the hour:minute:seconds, and YYYY is the year. For instance:

```
Wed Sep 04 15:13:22 1985\n\0
```

The time pointer returned by the function refers to a static data area that is shared by both ctime and asctime.

The time value argument t must point to a long integer that is the number of seconds since 00:00:00 Greenwich Mean Time, January 1, 1970. Normally this value is obtained from the time function. Note that ctime converts this value back into local time by calling _tzset and then subtracting the contents of timezone.

Note that t is a pointer to a time_t. A common error is to pass the time_t value itself instead of the pointer. Observe the use of the ampersand (&) operator in the following example.

SEE

asctime, gmtime, localtime, time, _tzset, utpack, utunpk

EXAMPLE

```
#include  <time.h>
#include  <stdio.h>

int  main(void)
{
  time_t  t;

  time(&t);
  printf("Current  time  is  %s\n",ctime(&t));
}
```

_CXFERR

Low-level float error exit

Class: Lattice

Category: Errors

SYNOPSIS

```
#include  <math.h>

_CXFERR(code);

int  code;
```

DESCRIPTION

The _CXFERR function is called when an error is detected by one of the low-level floating point routines, such as arithmetic operations. Higher-level routines, such as trigonometric functions, use the more sophisticated matherr.

Users can replace this error trap with an application-dependent routine, as long as they still store the error code in the global integer _FPERR. This is necessary because some of the maths functions check _FPERR to see if low-level errors occurred.

The error code passed to _CXFERR indicates the type of floating point anomaly that occurred, as follows, defined in math.h:

Symbol	Value	Meaning
FPEUND	1	Underflow
FPEOVF	2	Overflow
FPEZDV	3	Divide by zero
FPENAN	4	Not a number
FPECOM	5	Not comparable

SEE

matherr

_dclose — Close a GEMDOS file

Class: GEMDOS *Category: DOS Interface*

SYNOPSIS

```
#include <dos.h>

error = _dclose(fh);

long error;    0 for success, -1 for error
int fh;        file handle
```

DESCRIPTION

This function closes a GEMDOS file that was opened via _dcreat, _dcreatx or _dopen.

RETURNS

If the operation is successful, the function returns 0. Otherwise it returns -1 and places error information in errno and _OSERR.

SEE

Fclose, errno, _OSERR, _dcreat, _dcreatx, _dopen

_dcreat, _dcreatx — Create a GEMDOS file

Class: GEMDOS *Category: DOS Interface*

SYNOPSIS

```
#include <dos.h>

fh = _dcreat(name,fatt);     Create or truncate GEMDOS
                             file
fh = _dcreatx(name,fatt);    Create new GEMDOS file

long fh;                     file handle (-1 for error)
const char *name;            file name
int fatt;                    file attribute
```

DESCRIPTION

These functions create and open a GEMDOS file, returning the file handle. The _dcreat operation will truncate the file if it already exists, or create the file if it does not exist. Alternatively, _dcreatx will fail if the file already exists.

RETURNS

If the operation is successful, the function returns a file handle. Otherwise it returns -1 and places error information in errno and _OSERR.

SEE

Fcreate, errno, _OSERR, _dopen

dfind, dnext

Find directory entry

Class: GEMDOS

Category: DOS Interface

SYNOPSIS

```
#include  <dos.h>

err  =  dfind(info,name,attr);  Find  first  directory
                                 entry
err  =  dnext(info);             Find  next  directory
                                 entry

int  err;                        0  if  successful
struct  FILEINFO  *info;         file  information  area
const  char  *name;              file  name  or  pattern
int  attr;                       file  attribute  bits
```

DESCRIPTION

These functions search a directory for entries that match the specified file name or file name pattern. The dfind function locates the first matching file. Then successive calls to dnext locate additional matching files. Each dnext call must be given the file information that was returned on the preceding call to dfind or dnext.

The name argument must be a null-terminated string specifying the drive, path, and name of the desired file. The drive and path can be omitted, in which case the current directory will be searched. You can use the GEMDOS * and ? characters for pattern matching in the name portion. For example, xy*.b will locate files in the current directory that begin with xy and have b as their extension.

The attr argument specifies which file types are to be included in the search. The following bits are used:

Bit	Meaning
0	Read-only flag
1	Hidden file flag
2	System file flag
3	Volume label flag
4	Subdirectory flag

The info argument points to a file information structure as defined in the dos.h header file. For GEMDOS, this is the same as the GEMDOS DTA structure:

```
struct  FILEINFO
{
   char  resv[21];        /*  reserved  */
   char  attr;            /*  actual  file  attribute  */
   long  time;            /*  file  time  and  date  */
   long  size;            /*  file  size  in  bytes  */
   char  name[FNSIZE];    /*  file  name  */
};
```

RETURNS

If the operation is successful, a value of 0 is returned. Otherwise, the return value is -1, and further error information can be found in errno and _OSERR.

SEE

Fsfirst, Fsnext, getfnl, errno, _OSERR

EXAMPLE

```
/*
 *  show  the  files  in  a  given  directory
 */

#include  <dos.h>

void  showdir(const  char  *s)
{
  struct  FILEINFO  info;

  if  (!dfind(&info,name,0)
    do
    {
      puts(info.name);
    }  while  (!dnext(&info));
}
```

difftime

Compute difference between calendar times

Class: ANSI *Category: Date and Time*

SYNOPSIS

```
#include <time.h>

diff = difftime(time1,time0);

double diff;      difference between calendar times
                  (seconds)
time_t time1;     one calendar time
time_t time0;     another calendar time
```

DESCRIPTION

The difftime function computes the difference (in seconds) between two calendar times: time1 - time0. difftime was introduced as an ANSI function so that implementations could store an indication of the date/time value in the most efficient format possible and still provide a method of calculating the difference between two times.

RETURNS

This function returns the difference expressed in seconds as a double.

_disatty

Check if a GEMDOS handle is a terminal

Class: GEMDOS *Category: DOS Interface*

SYNOPSIS

```
#include <dos.h>

ret = _disatty(fh);

int ret;   0 if not a terminal
int fh;    file handle
```

DESCRIPTION

This function returns a non-zero value if the specified GEMDOS file handle is attached to a terminal (TTY) device, i.e. a console, printer or auxiliary device.

RETURNS

The return value is 0 if the file is not a terminal or if an error occurred while attempting to obtain the file's characteristics. You can check errno and _OSERR for detailed error information. If the file is a terminal, a value of 1 is returned.

SEE

isatty, errno, _OSERR

div, ldiv

Divide two signed integers

Class: ANSI *Category: Numeric Transformation*

SYNOPSIS

```
#include  <stdlib.h>

p  =  div(numer,denom);       Divide  two  signed  integers
q  =  ldiv(lnumer,ldenom)     Divide  two  signed  longs

div_t  p;                     quotient,  remainder
ldiv_t  q;                    long  quotient,  remainder
int  numer;                   numerator
int  denom;                   denominator
long  lnumer;                 long  numerator
long  ldenom;                 long  denominator
```

DESCRIPTION

The div and ldiv functions compute the quotient and remainder of the division of the numerator by the denominator. If the division is inexact, the resulting quotient is the integer of lesser magnitude that is the nearest to the algebraic quotient. The result can be represented as:

```
p.quot  *  denom  +  p.rem  =  numer
```

The div and ldiv functions provide a set of well-specified semantics for signed integral division and remainder operations. The semantics were adopted to be the same as FORTRAN. The following table summarises the semantics of these functions:

Numerator	Denominator	Quotient	Remainder
7	3	2	1
-7	3	-2	-1
7	-3	-2	1
-7	-3	2	-1

RETURNS

The div function returns a structure of type div_t, comprising both the quotient and the remainder, whilst the ldiv function returns a structure of type ldiv_t. The structures contain the following members:

```
int  quot;    /*  quotient  */
int  rem;     /*  remainder  */
```

_dopen

Open a GEMDOS file

Class: GEMDOS *Category: DOS Interface*

SYNOPSIS

```
#include  <dos.h>

fh  =  _dopen(name,mode);

long  fh;                 file  handle  (-1  for  error)
const  char  *name;       file  name
int  mode;                access  mode
```

DESCRIPTION

This function opens a GEMDOS file and returns the file handle. The mode argument must be a mode supported directly by GEMDOS, i.e. O_RDONLY, O_WRONLY and O_RDWR.

RETURNS

If the operation is successful, the function returns a file handle. Otherwise it returns -1 and places error information in errno and _OSERR.

SEE

Fopen, errno, _OSERR, open, _dcreat, _dcreatx, _dclose

drand

Generate random numbers

Class: UNIX *Category: Random Numbers*

SYNOPSIS

```
    #include <math.h>

    x = drand48();           random double (internal seed)
    x = erand48(seed);       random double (external seed)
    y = lrand48();           random positive long (internal
                             seed)
    y = nrand48(seed);       random positive long (external
                             seed)
    z = mrand48();           random long (internal seed)
    z = jrand48(seed);       random long (external seed)
    srand48(hseed);          set high 32 bits of internal
                             seed
    pseed = seed48(seed);    set all 48 bits of internal
                             seed
    lcong48(parm);           set linear congruence
                             parameters

    double x;                random double
    long y;                  random positive long
    long z;                  random long
    short seed[3];           seed value (high bits in
                             seed[0])
    long hseed;              high 32 bits of seed value
    short *pseed;            pointer to internal seed
    short parm[7];           parameters
```

DESCRIPTION

These functions generate various types of random numbers using the linear congruential algorithm and 48-bit arithmetic. The normal functions drand48, lrand48 and mrand48 use an internal 48-bit storage area for the seed value. Special versions erand48, jrand48 and nrand48 are provided for cases where several seeds are in use at the same time, in which case the user specifies the seed on each function call.

The drand48 and erand48 functions return double values distributed uniformly over the interval from 0.0 up to but not including 1.0.

The lrand48 and nrand48 functions return non-negative long integers uniformly distributed over the interval from 0 to 2**31-1.

The mrand48 and jrand48 functions return signed long integers uniformly distributed over the interval from -2**31 to 2**31-1.

The srand48 and seed48 functions allow initialisation of the internal 48-bit seed to something other than the default. For srand48 the specified long value is copied into the high 32 bits of the seed, and the low 16 bits are set to 0x330E. For seed48 the entire 48 bits are loaded from the specified array, and the function returns a pointer to the internal seed array.

The lcong48 function allows a much more intricate initialisation of the linear congruential algorithm. The algorithm is of the form:

```
    X[n+1] = (a * X[n] + c) mod m
```

where m is 2**48 and the default values for a and c are 0x5DEECE66D and 0xB, respectively. The array passed to lcong48 is structured as follows:

Parameter	Value
parm(0)	Bits 47-32 of value X(n)
parm(1)	Bits 31-16 of value X(n)
parm(2)	Bits 15-00 of value X(n)
parm(3)	Bits 47-32 of value a
parm(4)	Bits 31-16 of value a
parm(5)	Bits 15-00 of value a
parm(6)	value c

Whenever seed48 is called, a and c are reset to their default values.

RETURNS

As noted above.

SEE

rand, srand

_dread, _dwrite

Read and write GEMDOS files

Class: GEMDOS

Category: DOS Interface

SYNOPSIS

```
#include <dos.h>

cnt = _dread(fh,buf,len);     Read from a GEMDOS file
cnt = _dwrite(fh,cbuf,len);   Write to a GEMDOS file

long cnt;                     actual bytes read or
                              written
int fh;                       file handle
void *buf;                    data buffer
const void *cbuf;             data buffer
size_t len;                   number of bytes to read
                              or write
```

DESCRIPTION

These functions read or write a GEMDOS file whose handle was returned by _dcreat, _dcreatx or _dopen. Under normal circumstances, the value returned should match the buffer length. If this value is -1 or greater than the requested length, then some type of error occurred, and you should consult errno and _OSERR. If the actual length is less than the requested length when reading, this usually means that the file is exhausted. Similarly, if the actual length is less than the requested length for a write operation, this usually means that the device has no more space available. In both of these cases, it is still a good idea to check errno and _OSERR just in case some malfunction caused the short count.

RETURNS

If the operation is successful, the function returns the actual number of bytes transferred. Otherwise it returns -1 and places error information in errno and _OSERR.

SEE

errno, _OSERR, _dcreat, _dcreatx, _dopen, _dclose, _dseek

_dseek

Re-position a GEMDOS file

Class: GEMDOS

Category: DOS Interface

SYNOPSIS

```
#include <dos.h>

apos = _dseek(fh,rpos,mode);

long apos;       actual file position
int fh;          file handle
long rpos;       relative file position
int mode;        seek mode
```

DESCRIPTION

This function re-positions a GEMDOS file whose handle was returned by _dcreat, _dcreatx or _dopen. The seek mode is the same as for lseek as follows (defined in stdio.h):

Mode	Meaning
SEEK_SET	The rpos argument is the number of bytes from the beginning of the file. This value must be positive.
SEEK_CUR	The rpos argument is the number of bytes relative to the current position. This value can be positive or negative.
SEEK_END	The rpos argument is the number of bytes relative to the end of the file. This value must be negative or zero.

Note that for mode SEEK_CUR rpos can be positive or negative, but apos is always the actual (positive) position relative to the beginning of file.

RETURNS

If the operation is successful, the function returns the actual file position, which is a long integer. Otherwise it returns -1 and places error information in errno and _OSERR.

SEE

Fseek, errno, _OSERR, _dread, _dwrite

_ddup, _ddup2

Duplicate a GEMDOS file handle

Class: GEMDOS

Category: DOS Interface

SYNOPSIS

```
    #include  <dos.h>

    nfh  =  _ddup(fh);              Duplicate  a  file  handle
    error  =  _ddup2(nfh,fh);       Assign  a  file  handle

    int  nfh;                       new  file  handle
    int  fh;                        old  file  handle
    int  error;                     -1  if  error
```

DESCRIPTION

These functions duplicate a GEMDOS file handle. The new handle is associated with the same file as the old handle.

They are normally used in the same way as the higher level dup and dup2 functions for associating a different stdin, stdout, or stderr for a child process.

RETURNS

If the operation is successful, _ddup returns a file handle, while _ddup2 returns 0. Otherwise a value of -1 is returned, and error information is placed into errno and _OSERR.

Do not use these functions with files being accessed via open and the other low-level I/O functions. Use dup and dup2 instead.

SEE

Fdup, Fforce, dup, dup2, _dopen, _dclose, errno, _OSERR

dup, dup2

Duplicate a file handle

Class: UNIX

Category: Low-Level I/O

SYNOPSIS

```
    #include  <fcntl.h>

    nfh  =  dup(fh);                Duplicate  a  file  handle
    error  =  dup2(nfh,fh);         Assign  a  file  handle

    int  nfh;                       new  file  handle
    int  fh;                        old  file  handle
    int  error;                     -1  if  error
```

DESCRIPTION

These functions duplicate a file handle. The new handle is associated with the same file as the old handle.

Normally, dup is used when you want to establish a different stdin, stdout, or stderr for a child process. In order to preserve your current input, output, or error channel, you would use either dup or dup2 to duplicate file handle 0, 1, or 2. Then you would use fdopen to re-establish the association between the new handle and stdin, stdout, or stderr. Finally, you would open a file that you want to be the child process' standard input, output, or error channel; use dup2 if necessary to make the proper association with handle 0, 1, or 2.

RETURNS

If the operation is successful, dup returns a file handle, while dup2 returns 0. Otherwise a value of -1 is returned, and error information is placed into errno and _OSERR.

Do not use these functions with files being accessed via _dopen and the other low-level I/O functions. Use _ddup and _ddup2 instead.

SEE

Fdup, Fforce, _ddup, _ddup2, fdopen, errno, _OSERR

ecvt, fcvt

Convert float to string

Class: UNIX *Category: Data Conversion/Formatting*

SYNOPSIS

```
#include  <math.h>

s  =  ecvt(v,dig,decx,sign);  convert  float  to  string
s  =  fcvt(v,dec,decx,sign);  convert  float  to  string

char  *s;                     string  pointer
double  v;                    floating  point  value
int  dig;                     number  of  digits
int  dec;                     number  of  decimal  places
int  *decx;                   pointer  to  decimal  index
                              (returned)
int  *sign;                   pointer  to  sign  indicator
```

DESCRIPTION

These functions convert a floating point number into an ASCII character string consisting of digits only and terminated by a null character.

For ecvt, the second argument indicates the total number of digits that should be generated, while for fcvt it indicates how many digits should be generated to the right of the decimal place. If the floating point value contains fewer significant digits, zeroes are appended. If there are too many significant digits, the low order (right-most) digit is rounded.

The decx argument points to an integer that will receive a value indicating where the decimal point should be placed in the string. For example, an index value of 3 indicates that the decimal point should be placed just after the third character in the string. A value of zero means that the decimal point is just before the first character. If the index is negative, it indicates the number of zeroes that are between the decimal point and the first character. For example, -3 means that there are three zeroes between the decimal point and the beginning of the string.

The sign argument points to an integer that will be non-zero if v is negative.

EXAMPLE

```
#include  <math.h>

int  main(void)
{
  int  decx,sign;
  char  *string;

  string  =  ecvt(3.1415926535,10,&decx,&sign);
```

```
/*
 *  string  =>  "3141592654"
 *  decx  =>  1
 *  sign  =>  0
 */

  string  =  fcvt(3.1415926535,10,&decx,&sign);

/*
 *  string  =>  "31415926535"
 *  decx  =>  1
 *  sign  =>  0
 */
  return  0;
}
```

__emit

Emit 68000 instruction word

Class: Lattice

Category: Builtin Functions

SYNOPSIS

```
#include <dos.h>

__emit (x);

short x;  opcode to place in instruction stream
```

DESCRIPTION

The built-in function emit takes a constant 16-bit value corresponding to a 68000 assembly language instruction and inserts it in-line with the code. However, it does not check whether the 16-bit value is a valid 68000 instruction. It lacks the power and flexibility of an in-line assembler.

Note that this function is implemented as a macro expanding to the function __builtin_emit hence you *must* include the header file dos.h.

If one doesn't know how to use the emit function, it can create serious problems. While programmers may find this function useful in some situations, it should not be used without exercising a great deal of care and skill.

SEE

getreg, putreg

_end, _edata, _etext

Last locations in program

Class: UNIX

Category: Linker Defined Symbols

SYNOPSIS

```
extern __far _end;
extern __far _data;
extern __far _etext;
```

DESCRIPTION

These names refer to the last locations in the program. The address of _etext is the first location above the executable program text, that of _edata the first location above the initialised data area and _end the location immediately after the unitialised data area.

_ENEED

Maximum environment string space

Class: Lattice *Category: Process Environment*

SYNOPSIS

```
    extern int _ENEED;
```

DESCRIPTION

This external variable specifies the maximum number of environment strings which may be manipulated by the getenv, putenv and rmvenv commands. If it is smaller than that required for the process when it starts the value is ignored and the value allocated 4 times the number of strings available at startup.

environ

Strings forming user environment

Class: UNIX *Category: Process Environment*

SYNOPSIS

```
    extern char **environ;
```

DESCRIPTION

The external variable environ points to an array of strings forming the "environment". By convention these strings have the form "NAME=value". This array is normally manipulated by the functions getenv, putenv and rmvenv.

SEE

getenv, putenv, rmvenv, _ENEED

errno

UNIX error number

Class: ANSI

Category: Errors

SYNOPSIS

```
    #include <errno.h>

    extern  int  volatile errno;       UNIX error number
    extern  int  sys_nerr;             number of error codes
    extern  char  *sys_errlist[];      UNIX error messages
```

DESCRIPTION

The external integer named errno is initialised to 0 at start-up time. Then if an error is detected by one of the standard library functions, a non-zero value is placed there. The standard library never resets errno.

Programmers typically use this information in two ways. In some cases, it is appropriate to check errno after a sequence of operations and abort if any error occurred along the way. In other cases, errno is checked periodically, and if it is non-zero, the appropriate corrective action is taken. Then the application program resets errno before beginning the next processing phase.

The sys_nerr and sys_errlist items are defined in a C source file named syserr.c and are used by the perror function to print messages that correspond to the code found in errno. Note that the sys_ variables do *not* form part of the ANSI C standard.

Note that even though error information is normally placed into errno by the standard library functions, application programs can also use this technique to indicate problems. However, you should be careful about adding new codes and messages just above the highest UNIX code currently defined, since new UNIX codes are added occasionally. Also, we recommend that you add application-dependent codes by extending the header file errno.h, which contains symbolic definitions of the code numbers. The currently defined codes are listed as follows:

Symbol	Code	Meaning
EOSERR	-1	Operating system error
EPERM	01	User is not owner
ENOENT	02	No such file or directory
ESRCH	03	No such process
EINTR	04	Interrupted system call
EIO	05	I/O error
ENXIO	06	No such device or address
E2BIG	07	Argument list is too long
ENOEXEC	08	Exec format error
EBADF	09	Bad file number
ECHILD	10	No child process
EAGAIN	11	No more processes allowed
ENOMEM	12	No memory available
EACCES	13	Access denied
EFAULT	14	Bad address
ENOTBLK	15	Bulk device required
EBUSY	16	Resource is busy
EEXIST	17	File already exists
EXDEV	18	Cross-device link
ENODEV	19	No such device
ENOTDIR	20	Is not a directory
EISDIR	21	Is a directory
EINVAL	22	Invalid argument
ENFILE	23	No more files (system)
EMFILE	24	No more files (process)
ENOTTY	25	Not a terminal
ETXTBSY	26	Text file is busy
EFBIG	27	File is too large
ENOSPC	28	No space left
ESPIPE	29	Seek issued to pipe
EROFS	30	Read-only file system
EMLINK	31	Too many links
EPIPE	32	Broken pipe
EDOM	33	Math function argument error
ERANGE	34	Math function result is out of range

SEE

perror, strerror, sys_err

exit, _exit

Terminate program execution

Class: ANSI

Category: Process Creation

SYNOPSIS

```
#include <stdlib.h>

exit(code);  Terminate with clean-up
_exit(code); Terminate with no clean-up

int code;    status code
```

DESCRIPTION

These functions terminate execution of the current program and return control to the parent program. Use exit, for a graceful termination, which means that all pending output buffers are written and all files are explicitly closed. The _exit function terminates immediately without writing output buffers or closing files. Generally, this latter form is used only in emergency situations when you don't care if some output data is lost.

This function will normally be called after the code in main has been executed, and any return value from main is then passed to exit. Note that in general the _exit function is automatically called from the exit function after it has performed any clean up required.

In either case, the code is a value that gets passed back to the parent. By convention, a value of zero indicates success. If the parent is another C program that started this one up via one of the fork functions, then the parent can obtain the return code via the wait function.

RETURNS

This function does not return.

SEE

Pterm, Pterm0, onexit, atexit, forklpe, forkvpe, wait

EXAMPLE

```
/*
 * This example shows how you would abort a program
 * if it is not called with a valid input file name.
 *
 */
#include <stdio.h>
#include <stdlib.h>

int main(int argc,char *argv[])
{
  FILE *f;

  if(argc > 1)
  {
    f = fopen(argv[1],"r");
    if(!f)
    {
      fprintf(stderr,"Can't open file %s\n",argv[1]);
      return 1;
    }
  }
  else
  {
    fprintf(stderr,"No file specified\n");
    return 1;
  }

/*** Continue, now that file has been verified ***/
```

exp, et al

Exponential functions

Class: ANSI

Category: Mathematics

SYNOPSIS

```
#include <math.h>

r = exp(x);         exponential function
r = log(x);         natural logarithm function
r = log10(x);       base 10 logarithm function
r = pow(x,y);       power function
r = sqrt(x);        square root function
r = pow2(x);        compute 2**x

double r, x, y;
```

DESCRIPTION

The exp function raises the natural logarithm base e to the x power, and pow raises x to the y power. For pow, the x value must be an integer if it is negative. If it is not integral, matherr is called with a DOMAIN error.

The pow2 function computes 2^x by calling the pow function. The return value r is the value 2^x.

The log and log10 functions take the base e and base 10 logarithm, respectively. Each of these as well as sqrt, requires a positive argument. If a negative argument is supplied, matherr will be called with a DOMAIN error.

SEE

matherr

fabs

Absolute value of float/double

Class: ANSI

Category: Numeric Transformation

SYNOPSIS

```
#include <math.h>

ad = fabs(d);

double d;
double ad;
```

DESCRIPTION

The fabs function computes the absolute value of a float or a double, returning a double result.

SEE

abs, iabs, labs

fclose, fcloseall

Close a buffered file

Class: ANSI *Category: Stream I/O*

SYNOPSIS

```
    #include  <stdio.h>

    ret  =  fclose(fp);   close  a  buffered  file
    num  =  fcloseall();  close  all  buffered  files

    int  ret;             return  code
    int  num;             number  of  files  closed
    FILE  *fp;            file  pointer  for  file  to  be
                          closed
```

DESCRIPTION

The fclose function completes the processing of a buffered file (i.e. a file previously opened via fopen) and releases all related resources. The buffer associated with the file is released via the free function.

Even though fclose is automatically called for all open files when your program terminates or calls exit, it is good programming practice to close your own files explicity. The the last buffer is not written until fclose is called, and so data may be lost if an output file is not properly closed.

The fcloseall function closes all buffered files and returns the number of files that were closed. If an error occurs on any file, fcloseall continues to close the other files and then returns a value of -1.

RETURNS

Both functions return -1 to indicate an error. For success, fclose returns 0, and fcloseall returns the number of files that were closed. If -1 is returned, additional error information can be found in errno and _OSERR.

Remember that fcloseall closes the standard files stdin, stdout, and stderr. This means, for example, that functions such as printf and perror will fail after you call fcloseall.

SEE

fopen, errno, _OSERR

fdopen

Assign handle to buffered file

Class: UNIX *Category: Stream I/O*

SYNOPSIS

```
    #include  <stdio.h>

    fp  =  fdopen(fh,mode);

    FILE  *fp;            file  pointer
    int  fh;              file  handle
    const  char  *mode;   access  mode
```

DESCRIPTION

This function assigns a specific file handle to a buffered file. In other words, if you have used open to obtain a file handle, you can subsequently use buffered I/O with that file via fdopen. The mode argument for fdopen has the same form as for fopen.

RETURNS

If the operation is successful, the function returns a non-NULL file pointer. Otherwise it returns a NULL pointer and places error information in errno and _OSERR.

SEE

fopen, errno, _OSERR

feof

Check for end-of-file

Class: ANSI

Category: Stream I/O

SYNOPSIS

```
    #include  <stdio.h>

    ret  =  feof(fp);

    int  ret;   non-zero  if  end-of-file  is  found
    FILE  *fp;  file  pointer
```

DESCRIPTION

The feof function generates a non-zero value if the specified file is at end-of-file. Note that the specified file must have been opened previously via fopen or fdopen.

RETURNS

If an end-of-file is found, a non-zero value is returned.

This function is implemented as a macro, and does not check if fp is a valid file pointer.

SEE

ferror

ferror

Check for file error

Class: ANSI

Category: Stream I/O

SYNOPSIS

```
    #include  <stdio.h>

    ret  =  ferror(fp);

    int  ret;     non-zero  if  file  error  is  found
    FILE  *fp;    file  pointer
```

DESCRIPTION

The ferror function generates a non-zero value if an error has occurred on the specified file. Note that the file must have been opened previously via fopen or fdopen.

RETURNS

The return value is 0 if no error has occurred. If a file error has been found, a non-zero value is returned.

The ferror function is implemented as a macro, and does not check if fp is a valid file pointer.

SEE

feof

fflush, flushall

Flush file output buffer

Class: ANSI

Category: Stream I/O

SYNOPSIS

```
#include <stdio.h>

ret = fflush(fp);    Flush a file output buffer
num = flushall();    Flush all file output buffers

FILE *fp;            file pointer
int ret;             return code
int num;             number of open files
```

DESCRIPTION

The fflush macro flushes the output buffer of a file previously opened via fopen or fdopen. That is, it writes the buffer if the file is opened for output and the buffer contains any pending data. If an error occurs, the return value is EOF and the appropriate error code is placed into errno.

The flushall function flushes all file output buffers and returns the number of files that are open. If an error occurs, the function continues to flush the remaining files and then returns a value of -1.

RETURNS

As noted above. In the event of a -1 return, error information can be found in errno and _OSERR.

SEE

fopen, fclose, errno, _OSERR

fgetc, fgetchar

Get a character

Class: ANSI

Category: Stream I/O

SYNOPSIS

```
#include <stdio.h>

c = fgetc(fp);      Get a character from a file
c = fgetchar();     Get a character from stdin

int c;              return character or code
FILE *fp;           file pointer
```

DESCRIPTION

These functions get a single character from a file that was previously opened via fopen or fdopen. For fgetchar, the standard input file is read via file pointer stdin.

RETURNS

Upon success, the next input character is returned. Otherwise, the functions return EOF, which is defined in stdio.h.

In the event of an EOF return, error information can be found in errno and _OSERR. Most programmers treat any EOF return as an indication of end-of-file. However, if you want to distinguish errors from end-of-files, you should reset errno before calling the function and then analyse its contents when you receive an EOF return.

SEE

errno, fopen, getc, getchar, _OSERR

fgetpos — Store current value of file position indicator

Class: ANSI *Category: Stream I/O*

SYNOPSIS

```
    #include  <stdio.h>

    ret  =  fgetpos  (strm,pos);

    int  ret;            0  if  successful
    FILE  *strm;         stream
    fpos_t  *pos;        file  position  info
```

DESCRIPTION

The fgetpos function stores the current value of the file position indicator for the stream pointed to by stream in the object pointed to by pos. The value stored in pos contains information usable by the fsetpos function for repositioning the stream to its position at the time of the call to the fgetpos function.

RETURNS

If successful, the fgetpos function returns 0; on failure, the fgetpos function returns non-zero and stores an the error value in errno.

SEE

fsetpos

fgets — Get a string from a buffered file

Class: ANSI *Category: Stream I/O*

SYNOPSIS

```
    #include  <stdio.h>

    p  =  fgets(buffer,length,fp);

    char  *p;        buffer  pointer  or  NULL
    char  *buffer;   buffer  pointer
    int  length;     buffer  length  in  bytes
    FILE  *fp;       file  pointer
```

DESCRIPTION

The fgets function gets a string from the specified file, which must have been previously opened for input via fopen or fdopen. Characters are copied from the file to the buffer until a newline ('\n') has been copied, or the buffer is full, or the end-of-file is hit. In the newline case, a null byte ('\0') is placed into the buffer after the newline if the buffer has room. In the end-of-file case, a null byte is placed into the buffer after the last byte that was read. If the end-of-file is hit before any bytes are read, a NULL pointer is returned.

Note that the returned string will not be null-terminated if length characters have already been placed into the buffer.

RETURNS

The fgets function returns the buffer argument unless an end-of-file or I/O error occurs, in which case a NULL pointer is returned.

SEE

errno, feof, ferror, fgetc, fopen, getc, gets

EXAMPLE

```
/*
 *  Assume that stdin contains the following lines:
 *
 *  Hello, folks!
 *  Goodbye, folks!
 *  (blank line or EOF)
 */
#include <stdio.h>

  char *p,b[80];
  /* For the next two lines, p will point to b */
  p = gets(b);
  /* Now b contains "Hello, folks!" */
  p = fgets(b,sizeof(b),stdio);
  /* Now b contains "Goodbye, folks!\n" */
  p = gets(b);
  /* Now p is NULL */
```

fgetw, fgetl

Get a word/longword from a buffered file

Class: UNIX *Category: Stream I/O*

SYNOPSIS

```
#include <stdio.h>

x = fgetw(fp);
y = fgetl(fp);

short x;        word value from stream
long y;         longword value from stream
FILE *fp;       file pointer
```

DESCRIPTION

The fgetw and fgetl functions read words and longwords respectively from the associated file. If end-of-file is reached, EOF cast to the appropriate type is returned. Note that it may not be possible to distinguish EOF from legitimate characters and so the value of feof should be checked in these cases.

Note that these functions produce files which are highly non-portable as they give no indication of the ordering of bytes on the machines architecture.

RETURNS

The functions return a value from the stream or the value EOF if an end-of-file or I/O error occurs.

SEE

errno, feof, ferror, fgetc, fread, fputw, fputl

filelength

Find length of an unbuffered file

Class: Microsoft *Category: Low-Level I/O*

SYNOPSIS

```
#include  <fcntl.h>

length  =  filelength(fh);

long  length;   length  of  file  in  bytes  or  -1
int  fh;        unbuffered  file  handle
```

DESCRIPTION

The filelength function calculates the size of the file associated with the unbuffered file handle fh. The file handle should be one which was returned by an open or creat call.

RETURNS

The filelength function returns the number of bytes in the file, or if an error occurs returns -1 and sets errno accordingly.

SEE

creat, fileno, open

EXAMPLE

```
/*
 *  Find  the  length  of  a  buffered  file
 */
#include  <stdio.h>
#include  <fcntl.h>

long  len(FILE  *fp)
{
  fflush(fp);  /*  flush  any  buffered  bytes  to  disk  */
  return  filelength(fileno(fp));
}
```

fileno

Get handle for buffered file

Class: UNIX *Category: Stream I/O*

SYNOPSIS

```
#include  <stdio.h>

fh  =  fileno(fp);

int  fh;     file  handle
FILE  *fp;   file  pointer
```

DESCRIPTION

This function returns the file handle (i.e. the file number) associated with the specified file pointer. The file pointer must be one that was returned by fopen, freopen, or fdopen.

RETURNS

As noted above.

This function is implemented as a macro, and it does not check that fp is a valid file pointer.

_fmask

Set default protection mode for buffered I/O

Class: Lattice

Category: Stream I/O

SYNOPSIS

```
    extern long _fmask;
```

DESCRIPTION

This external integer is used by the fopen function to determine the protection mode to use when creating buffered files. The default is the value S_IWRITE | S_IREAD, giving both read and write privileges to any file created.

SEE

fopen

fmod

Compute floating point modulus

Class: ANSI

Category: Numeric Transformation

SYNOPSIS

```
    #include <math.h>

    x = fmod(y,z);

    double x;          floating point modulus
    double y;          dividend
    double z;          divisor
```

DESCRIPTION

The fmod function computes the floating point remainder of y/z. It returns y if z is 0. Otherwise, it returns a value that has the same sign as y, is less than z, and satisfies the relationship:

```
    y = (i * z) + x
```

where i is an integer. This is, in effect, what the expression:

```
    x = y % z;
```

would produce if the % operator were defined for floating point numbers.

SEE

modf

EXAMPLE

```
    #include <math.h>

    double r,ff,fi;

    r = fmod(5.7,1.5);      /* r contains 1.2 */

    ff = modf(r,&fi);       /* ff contains 0.2 */
                            /* fi contains 1.0 */
```