time.h — Date and Time manipulation functions

Class: ANSI *Category: Date and Time*

SYNOPSIS

```
typedef ... time_t;   type returned by time()
typedef ... clock_t;  type returned by clock()

const int CLK_TCK;         clock() granularity
const int CLOCKS_PER_SEC; clock() granularity

struct tm
{
  int tm_sec;      /* seconds after the minute */
  int tm_min;      /* minutes after the hour */
  int tm_hour;     /* hours since midnight */
  int tm_mday;     /* day of the month */
  int tm_mon;      /* months since January */
  int tm_year;     /* years since 1900 */
  int tm_wday;     /* days since Sunday */
  int tm_yday;     /* days since January 1 */
  int tm_isdst;    /* Daylight Savings Time flag */
};

clock_t clock(void);
double difftime(time_t, time_t);
time_t mktime(struct tm *);
time_t time(time_t *);
char *asctime(const struct tm *);
char *ctime(const time_t *);
struct tm *gmtime(const time_t *);
struct tm *localtime(const time_t *);
size_t strftime(char *, size_t, const char *,
                                const struct tm *);

void getclk(unsigned char *);
int  chgclk(unsigned char *);
void utunpk(long, char *);
long utpack(const char *);

void _tzset(void);

extern int   __daylight;   daylight time flag
extern long  __timezone;   seconds from GMT
extern char *__tzname[2]; time zone names
extern char  __tzstn[4];   standard time name
extern char  __tzdtn[4];   daylight time name

extern char *_TZ;     string for user time zone
```

DESCRIPTION

The time.h header file contains functions and macros for manipulating time in both internal and external representations.

Note that although ANSI defines this header file the getclk, chgclk, utunpk and utpack do not appear as part of the standard.

3 Library Functions

This section gives detailed descriptions of the library functions supplied with the Lattice C compiler, listing the header file in which the function is declared, the calling syntax and any parameters which should be supplied to the function.

As mentioned earlier, each entry consists of a synopsis, description and cross-reference. Also a 'Class' and 'Category' are listed giving the origin of the function, e.g. ANSI, Lattice, UNIX etc., and a category showing which family of functions the function falls into, e.g. Stream I/O, Date and Time.

In the past many C programmers have neglected to include the required header files and simply placed a declaration in their own file. This practice is strongly discouraged as ANSI changed the types of the parameters of many functions from the default int, hence your code may not run successfully without in-scope prototypes.

abort

Abort the current process

Class: ANSI

Category: Process Creation

SYNOPSIS

```
#include <stdlib.h>

abort();
```

DESCRIPTION

This function aborts the current process and returns a completion code of 3 to the parent process. Also the message "Abnormal program termination" is sent to stderr. I/O buffers created via fopen are not flushed. Prior to termination the signal SIGABRT is asserted, as if by the call:

```
raise(SIGABRT);
```

RETURNS

The function does not return.

SEE

onexit, exit, _exit, raise

EXAMPLE

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

void validate(int x,int lower,int higher)
{
  if (x<lower || x>higher)
  {
    puts("Internal range check failed");
    abort();
  }
}
```


abs

Absolute value

Class: ANSI

Category: Numeric Transformation

SYNOPSIS

```
#include <stdlib.h>

ax = abs(x);

int x;      numeric data type
int ax;     absolute value of x
```

DESCRIPTION

The abs function computes the absolute value of the integer argument. Compare abs with the fabs function, which computes the absolute value of a float or a double, returning a double result.

Note that this function is normally implemented as the inline function __builtin_abs.

SEE

fabs, iabs, labs

access — Check file accessibility

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

SYNOPSIS

```
#include <stdio.h>

ret = access(name,mode);

int ret;            return code
const char *name;   file name
int mode;           access mode
```

DESCRIPTION

This function checks if a file is accessible in the way specified by mode, which follows the UNIX format:

0	Check if file exists
2	Check if file is writable
4	Check if file is readable
6	Check if file is readable and writable

The other access mode bits recognised by UNIX are not supported under GEMDOS. Also, since all GEMDOS files are readable, modes 0 and 4 are identical, as are modes 2 and 6.

RETURNS

A return value of 0 indicates that access is allowed. If access is denied or the file cannot be found, -1 is returned. Additional error information can then be found in errno and _OSERR.

SEE

chgfa, getfa, errno, _OSERR

alloca — Allocate temporary stack space

Class: UNIX *Category: Memory Management*

SYNOPSIS

```
#include <stdlib.h>

s = alloca(n);

void *s;        pointer to base of memory
size_t n;       number of bytes required
```

DESCRIPTION

The alloca function obtains the specified number of bytes from the program's stack space. The value n gives the number of bytes required, and the return pointer s points to an area of the size requested, or NULL if insufficient stack is available.

Note that you should not attempt to return the space allocated via alloca using the free call. Any space allocated using this function is automatically reclaimed on function exit.

RETURNS

The value s is NULL if no more stack is available.

SEE

calloc, free, malloc, realloc

EXAMPLE

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

FILE *newfile(const char *s)
{
  char *p;

  p=alloca(strlen(s)+5);
  if (!p)
    return NULL;
  strcpy(p,s);
  strcat(p,".tmp");
  return fopen(p,"rb");
}
```

argopt

Get options from argument list

Class: Lattice

Category: Argument Processing

SYNOPSIS

```
#include <stdlib.h>

optd = argopt(argc,argv,opts,argn,optc);

char *optd;             option data pointer
int argc;               argument count
const char *argv[];     argument vector
const char *opts;       options expecting data
int *argn;              next argument number (changed)
char *optc;             option character (changed)
```

DESCRIPTION

This function examines an argument list to find the next option argument, using conventions similar to those of the UNIX "shell" command processor. These conventions are:

- An option is an argument that begins with a slash (/) or a dash (i.e. a minus sign) and appears between the command verb (i.e. argv(0)) and the first non-option argument. The reason we recognise either a slash or a dash is that the former is an MS-DOS standard, while the latter has been used by UNIX for a long time.
- The character immediately following the dash is called the "option character", and it may be followed by a character string known as the "option data".
- If the option character appears in the opts string, then the data can be separated from the character by white space. In effect, this means that the data might be in the next argv entry if it does not follow the option character in the current entry.
- A dash or slash followed by a blank or a dash indicates the end of the options.

Each time argopt is called, it will find the next option in the argument array and update the integer referenced by argn. On the first call, you should set this integer to 1, since argv(0) points to the command verb. The argc and argv items are normally the same as those passed to your main program, and they are not changed as a result of the argopt calls. The option character is returned in the byte referenced by optc, and the function returns a pointer to the option data string or to a null byte. If the next entry in argv is not an option, then the function returns a NULL pointer.

The opts item provides some flexibility in the way the option data is handled. If opts points to an empty string, then any option data must immediately follow the option character. However, if opts is not empty, then it lists the option characters that always have data. For those characters, the data can be preceded by white space on the command line. What this actually means is that argopt will look at the next entry in argv if the option character is not followed by a data string. If the next entry does not begin with a dash, then it is taken as the option data.

RETURNS

If the next argument is not an option, the function returns a NULL pointer. Otherwise, it returns a pointer to the option data, which will be an empty string if there was no data. If an option was found, the character is placed into the byte referenced by optc, and argn is adjusted to index the next entry in argv.

SEE

getopt, main

EXAMPLE

```
/*
 * Assume that this program is invoked by the
 * following command line:
 *
 *   myprog -x -ypdq -z -g moo -g - blah
 *
 * The output will then be:
 *   Option: x Data:
 *   Option: y Data: pdq
 *   Option: z Data:
 *   Option: g Data: moo
 *   Option: g Data:
 *   Arg[8]: blah
 *
 */

#include <stdio.h>
#include <stdlib.h>

char opts[] = "gx";

int main(int argc,char *argv[])
{
  char option,*odata;
  int next;

  for(next = 1;
    odata = argopt(argc,argv,opts,&next,&option); )
    printf("Option: %c, Data: %s\n",option,odata);

  for (; next < argc; next++)
    printf("Arg[%d]: %s\n",next,argv[next]);
  return 0;
}
```

asctime

Generate ASCII time string

Class: ANSI

Category: Date and Time

SYNOPSIS

```
#include <time.h>

s = asctime(t);

char *s;                points to time string
const struct tm *t;     points to time structure
```

DESCRIPTION

This function converts a time structure into 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 or asctime. The time structure argument t is usually returned by the gmtime or localtime function.

SEE

ctime, gmtime, localtime, setlocale

EXAMPLE

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

int main(void)
{
  struct tm *tp;
  time_t t;

  time(&t);
  tp = localtime(&t);
  printf("Current time is %s\n",asctime(tp));
  return 0;
}
```


assert

Assert program validity

Class: ANSI

Category: Debugging

SYNOPSIS

```
#include <assert.h>

assert(exp);

int exp;            expression to be tested
```

DESCRIPTION

The assert macro tests an expression exp for validity (non-zero value). Note that the assert.h header file must be included in your program in order to define the macro. If the expression being tested fails (i.e. is zero) then the program is aborted printing the text of the failing expression, file and line number on stderr.

Also, assert.h contains two versions of the macro. If the symbol NDEBUG is defined, then a null version of the macro is used; otherwise the normal code-generating version applies. This allows you to strip the assertion code from your program without removing the assert calls. To do this, simply define NDEBUG in one of your header files or on the compiler command line via the -d option. In the former case, the header file containing the NDEBUG definition must be included before assert.h.

EXAMPLE

```
/* Make sure integer x is positive */

#include <assert.h>

void postest(int x)
{
  assert(x >= 0);
}
```

atexit — Register function

Class: ANSI *Category: Process Creation*

SYNOPSIS

```
#include <stdlib.h>

ret = atexit((*func)())

int ret;            0 if successful
void (*func)(void); function to be registered
```

DESCRIPTION

The atexit function registers the function pointed to by func, to be called without arguments at normal program termination. The atexit function provides a program with a convenient way to clean up the environment before the program exits. It provides a last-in first-out stacking of multiple functions. The chain of registered functions is maintained in such a way that they are invoked in the correct sequence upon program exit.

The functions registered by atexit are invoked before any files are closed or memory is freed. The SIGTERM signal is raised before atexit.

RETURNS

The atexit function returns 0 if the registration succeeds, and non-zero if it fails to allocate memory for its list.

SEE

exit, onexit

atof — Convert ASCII to float

Class: ANSI *Category: Data Conversion/Formatting*

SYNOPSIS

```
#include <stdlib.h>

d = atof(p);

double d;       floating point result
const char *p;  input string pointer
```

DESCRIPTION

The atof function converts an ASCII input string into a double value. The string can contain leading white space and a plus or minus sign, followed by a valid floating point number in normal or scientific notation. If scientific notation is used, there can be no white space between the number and the exponent. For example:

```
123.456e-53
```

is a valid number in scientific notation.

EXAMPLE

```
/*
 * This program tests the atof function.
 */

#include <stdio.h>
#include <stdlib.h>

int main(void)
{
  char buff[80];
  double d;

  for (;;)
  {
    printf("\nEnter a number: ");
    if(gets(buff) == NULL)
      exit(0);
    if(buff[0] == '\0')
      exit(0);
    d = atof(buff);
    printf("%e\n",d);
  }
  return 0;
}
```

atoi, atol

Convert ASCII to integer

Class: ANSI

Category: Data Conversion/Formatting

SYNOPSIS

```
#include <stdlib.h>

x = atoi(s);        Convert ASCII to integer
y = atol(s);        Convert ASCII to long integer

int x;              integer result
long int y;         long integer result
const char *s;      input string pointer
```

DESCRIPTION

These functions convert ASCII strings into normal or long integers. The string must have the form:

```
[whitespace][sign]digits
```

where (whitespace) indicates optional leading white space, (sign) indicates an optional + or - sign character, and digits is a continuous string of digit characters. Once the digit portion is reached, the conversion continues until a non-digit character is hit. No check is made for integer overflow.

RETURNS

As noted above.

SEE

atof, stcd_i, stcd_l, strtol, strtoul

_base

Base of stack

Class: Lattice

Category: Process Environment

SYNOPSIS

```
extern void *_base;
```

DESCRIPTION

This external pointer is used by the stack check code to locate the base of the stack. If the stack pointer is in danger of overrunning this then the function _xcovf is called.

SEE

_STACK, _xcovf

bldmem

Build a memory pool of specified size

Class: OLD

Category: Memory Management

SYNOPSIS

```
#include <stdlib.h>

bldmem(n);

int n; number of 1K-byte blocks in pool
```

DESCRIPTION

The bldmem function builds up to n contiguous 1K-byte blocks of memory for the pool. If n is 0, the pool is initialised but no memory is allocated.

RETURNS

Returns -1 if memory cannot be allocated.

SEE

getmem, getml, rlsmem, rlsml, sizmem, sbrk

bsearch

Search a data array

Class: ANSI

Category: Search and Sort

SYNOPSIS

```
#include <stdlib.h>

match=bsearch(key,base,num_mem,size,(*cmp)(obj,arr));

void *match;          matched element or NULL pointer
const void *key;      object to be matched
const void *base;     initial element of searched array
size_t num_mem;       size of array to be searched
size_t size;          size of each element
int (*cmp)();         comparison function
const void *obj;      pointer to key
const void *arr;      pointer to an array element
```

DESCRIPTION

The bsearch function searches an array of num_mem objects (the initial element of which is pointed to by base) for an element that matches the object pointed to by key. The size of each element of the array is specified by size.

The comparison function pointed to by cmp is called with two arguments that point to the key object and to an array element, in that order. The function returns an integer less than, equal to, or greater than zero if the key object is considered, respectively, to be less than, to match, or to be greater than the array element. The array consists of all the elements that compare less than the key object, all the elements that compare equal to the key object, and all the elements that compare greater than the key object, in that order.

RETURNS

The bsearch function returns a pointer to a matching element of the array, or a NULL pointer if no match is found. If two elements compare as equal, the element matched could be either one.

SEE

lfind, lsearch

_BSSBAS, _DATABAS — Base of merged data sections

Class: Lattice *Category: Linker Defined Symbols*

SYNOPSIS

```
extern __far _BSSBAS;
extern __far _DATABAS;
```

DESCRIPTION

These names refer to the base locations in the __MERGED data section. The location of _BSSBAS is the first byte of the merged BSS, whilst _DATABAS is the first byte of the merged data.

SEE

_BSSLEN, _DATALEN

_BSSLEN, _DATALEN — Merged section lengths

Class: Lattice *Category: Linker Defined Symbols*

SYNOPSIS

```
extern __far _BSSLEN;
extern __far _DATALEN;
```

DESCRIPTION

These addresses of these names give the length of the respective __MERGED data section in *longwords*. Note that if you access these variables from assembly language you *must* access them as longs otherwise the assembler may attempt to relocate them, giving random values as a result.

SEE

_BSSBAS, _DATABAS

EXAMPLE

```
/*
 * Clear out the merged BSS in a program
 *
 * Normally done automatically
 */

#include <string.h>

int main(void)
{
  extern __far _BSSBAS;
  extern __far _BSSLEN;

  memset(&_BSSBAS,0,(long)&_BSSLEN/sizeof(long));
  ...
  return 0;
}
```

_bufsiz

Buffered I/O buffer size

Class: Lattice

Category: Stream I/O

SYNOPSIS

```
extern int _bufsiz;
```

DESCRIPTION

This external integer is used by the buffered I/O system to determine the size of the buffers for buffered files. This location is also used to determine the size of a buffer attached to a file with the setbuf function. In this case, _bufsiz must be set to the size of the buffer before setbuf is called.

Note that the buffer is not allocated when the file is opened. Instead, the first I/O operation causes the buffer to be allocated from the local memory pool if one has not been previously specified with setbuf. This means that if _bufsiz is changed between the open call and the first I/O operation, the size of the buffer allocated for the file will be the value of _bufsiz at the time of the I/O operation, not the value when the file was opened.

SEE

fopen, setbuf, setvbuf

cabs

Absolute value of a complex number

Class: Lattice

Category: Mathematics

SYNOPSIS

```
#include <math.h>

r = cabs(x);

double r;

struct complex {
  double re;
  double im;
} *x;
```

DESCRIPTION

The cabs function calculates the absolute value of a complex number pointed to by x. cabs(x) returns the value sqrt(x->re * x->re + x->im * x->im). If an overflow occurrs, matherr is called with an OVERFLOW error and suggested return value of HUGE_VAL.

cadd, csub

Complex sum/difference

Class: Lattice

Category: Mathematics

SYNOPSIS

```
#include  <math.h>

z  =  cadd(x,y,z);
z  =  csub(x,y,z);

struct  complex  {
  double  re;
  double  im;
} *z;

struct  complex  *x,  *y;
```

DESCRIPTION

The cadd function calculates the complex sum of the complex numbers pointed to by x and y, and places the result in the complex number pointed to by z. The pointer z is returned by the function.

Similarly csub calculates the complex difference of the numbers pointed to by x and y, and places the result in the complex number pointed to by z. The pointer z is returned by the function.

For instance, the expression:

```
z  =  cadd(x,y,z);
```

produces the following assigments:

```
z->re  =  x->re  +  y->re;
z->im  =  x->im  +  y->im;
```

Whilst the expression:

```
z  =  csub(x,y,z);
```

produces the following assigments:

```
z->re  =  x->re  -  y->re;
z->im  =  x->im  -  y->im;
```


calloc

Allocate and clear a memory block

Class: ANSI

Category: Memory Management

SYNOPSIS

```
#include  <stdlib.h>

b  =  calloc(nelt,esize);

void  *b;          block  pointer
size_t  nelt;      number  of  elements
size_t  esize;     element  size
```

DESCRIPTION

The calloc function uses malloc to get a block whose size in bytes is given by:

```
n  =  nelt  *  esize;
```

The block is then cleared to zeroes. Like malloc, calloc returns a NULL pointer if the block cannot be allocated.

RETURNS

The calloc function call normally returns a pointer to the block. If there is not enough space for the requested block, or if zero bytes are requested, a NULL pointer is returned.

SEE

free, getmem, malloc, realloc, rlsmem, rbrk, sbrk

cdiv — Complex quotient

Class: Lattice *Category: Mathematics*

SYNOPSIS

```
#include <math.h>

z = cdiv(x,y,z);

struct complex {
  double re;
  double im;
} *z;

struct complex *x, *y;
```

DESCRIPTION

The cdiv function calculates the complex quotient of complex numbers pointed to by x and y, and places the result in the complex number pointed to by z. The pointer z is returned by the function.

For instance, the expression:

```
z = cdiv(x,y,z)
```

produces the assignments:

```
z->re = ( x->re * y->re + x->im * y->im ) /
                (y->re * y->re + y->im * y->im);

z->im = ( x->im * y->re - x->re * y->im ) /
                (y->re * y->re + y->im * y->im);
```


ceil, floor — Get floating point limits

Class: ANSI *Category: Mathematics*

SYNOPSIS

```
#include <math.h>

x = ceil(y);  Get ceiling of a real number
x = floor(y); Get floor of a real number

double x,y;
```

DESCRIPTION

These functions return the integral values that are nearest to the specified real number. For ceil, the return is the next higher integer, while floor returns the next lower integer.

Note that although these functions return integral values, the results are still real numbers.

EXAMPLE

```
#include <math.h>

double r;

r = ceil(523.96);     /* r contains 524.0 */
r = floor(523.96);    /* r contains 523.0 */
```

cget, cgetc, cgets

Console input operations

Class: Lattice

Category: Console and Port I/O

SYNOPSIS

```
#include <dos.h>

c = cget();          get character from console,
                     no echo
c = cgetc();         get character from console, echo
p = cgets(buffer);   get string from console

int c;               input character
char *buffer;        input buffer
char *p;             input buffer
```

DESCRIPTION

These functions get single characters or character strings from the console keyboard. The cget and cgetc functions are equivalent to getch and getche, respectively. Also, cgetc and cgets are similar to getchar and gets, respectively. The console functions use the low-level keyboard routines directly rather than working through the file manager. This can result in improved performance in a highly interactive application.

RETURNS

If c is zero, then cget should be called again to obtain the keyboard scan code. This will happen when the user presses a key that cannot be translated into an ASCII code; e.g. a function key. The return from cgets is the buffer pointer.

SEE

cscanf, getch, getche, gets, kbhit

chdir

Change current directory

Class: UNIX

Category: Process Environment

SYNOPSIS

```
#include <stdio.h>

error = chdir(path);

int error;           0 if successful
const char *path;    points to new directory path
                     string
```

DESCRIPTION

This function changes the current directory to the specified path. Under GEMDOS, the path may begin with a drive letter and a colon.

RETURNS

If the return value is non-zero, then the operation failed. A GEMDOS error code will be in _OSERR, and a UNIX error code will be in errno.

SEE

Dsetpath, mkdir, rmdir, getcd, getcwd

chgclk — Change system clock

Class: Lattice *Category: DOS Interface*

SYNOPSIS

```
#include <dos.h>

error = chgclk(clock);

int error;
const unsigned char *clock;
```

DESCRIPTION

The chgclk function changes the setting of the system clock, using the following 8-byte array:

Byte	Contents
0	Day of week (0 for Sunday)
1	Year - 1980
2	Month (1 to 12)
3	Day (1 to 31)
4	Hour (0 to 23)
5	Minute (0 to 59)
6	Second (0 to 59)
7	Hundredths (0 to 99)

RETURNS

If the array is invalid, chgclk returns a non-zero value. In that case, the system clock may be partially changed under GEMDOS, since the date and time are updated on separate GEMDOS calls, either of which may have failed.

If your machine is equipped with a hardware clock, its state is not necessarily changed by a call to chgclk.

SEE

Tsetdate, Tsettime, errno, getclk, _OSERR

chgdsk, getdsk — Change or get current disk drive

Class: GEMDOS *Category: Disk Functions*

SYNOPSIS

```
#include <dos.h>

bmap = chgdsk(drive);
drive = getdsk();

int drive;      drive code
int bmap;       bitmap of mounted drives
```

DESCRIPTION

The chgdsk function changes the current drive code. Drive code 0 corresponds to drive A, code 1 is drive B and so on.

The getdsk function gets the current drive code, using the same codes as chgdsk.

RETURNS

The function chgdsk returns a bitmap of mounted drives, bit 0 corresponds to drive A, bit 1 is drive B and so on.

The function getdsk returns the code of the currently selected drive.

SEE

Dsetdrv, Dgetdrv, getcd

chgdta, getdta — Set/Get data transfer address (DTA)

Class: GEMDOS *Category: DOS Interface*

SYNOPSIS

```
#include <dos.h>

chgdta(dta);
dta = getdta();

struct FILEINFO *dta;   pointer to new DTA
```

DESCRIPTION

The chgdta function is used to change the data transfer address used by GEMDOS in the Fsfirst and Fsnext calls. By comparison the getdta function returns the current data transfer address.

SEE

Fsetdta, Fgetdta, Fsfirst, Fsnext, dfind, dnext

chgfa — Change file attribute

Class: GEMDOS *Category: File System Manipulation*

SYNOPSIS

```
#include <dos.h>

error = chgfa(name,fa);

int error;              0 if successful
int fa;                 file attribute
const char *name;       file name
```

DESCRIPTION

This function sets the attribute byte for the specified file. The attributes in fa are:

Bit	Meaning
0	Read-only flag
1	Hidden file flag
2	System file flag
3	Volume label flag
4	Subdirectory flag
5	Archive flag (set if file has changed)
6	Reserved
7	Reserved

Note that the archive bit is only supported correctly in version 1.4 and above of the operating system.

RETURNS

If the operation is unsuccessful, the function returns -1 and places error information in errno and _OSERR.

SEE

Fattrib, chmod, getfa, errno, _OSERR

chgft

Set file time

Class: GEMDOS

Category: File System Manipulation

SYNOPSIS

```
#include <dos.h>

error = chgft(fh,ft);

int error;   0 if successful
long ft;     file time
int fh;      file handle
```

DESCRIPTION

This function sets the time and date information associated with the specified file. This information usually indicates when the file was created or last updated. It has the following format:

Bits	Contents
00-04	Second/2 (0 to 29)
05-10	Minute (0 to 59)
11-15	Hour (0 to 23)
16-20	Day (0 to 31)
21-24	Month (1 to 12)
25-31	Year-1980 (0 to 127)

RETURNS

The chgft function returns 0 if successful or a value of -1 if in error. Additional error information can be found in errno and _OSERR.

SEE

Fdatime, getft, errno, _OSERR

chkml

Check for largest memory block

Class: OLD

Category: Memory Management

SYNOPSIS

```
#include <stdlib.h>

size = chkml();

long size;
```

DESCRIPTION

This function returns the size, in bytes, of the largest block that is currently available without calling upon the operating system to supply additional heap space.

SEE

getmem, getml, rlsmem, rlsml, sizmem

chkufb

Check unbuffered file handle

Class: Lattice

Category: Low-Level I/O

SYNOPSIS

```
#include <ios1.h>

ufb = chkufb(fh);

struct UFB *ufb;  pointer to UNIX file block
int fh;           file handle
```

DESCRIPTION

This function checks if a file handle is currently associated with an unbuffered file. Normally it is used internally by open, close, read, write, lseek and tell. The UFB structure is defined in header file ios1.h. For GEMDOS this structure is two short integers. The first contains the mode flags specified in the call to the open function. The second contains the file handle. The external name _ufbs refers to an array of UFB structures, and the external integer _nufbs indicates how many structures are in the array. Normally this value is fourty.

RETURNS

If no UFB is currently attached to the file handle, a NULL pointer is returned.

chmod

Change file protection mode

Class: UNIX

Category: File System Manipulation

SYNOPSIS

```
#include <stdio.h>

error = chmod(name,mode);

int error;          error code
const char *name;   file name
int mode;           protection mode
```

DESCRIPTION

This function changes a file's protection mode. It is compatible with UNIX, although GEMDOS provides only a single write-protection bit for each file. The mode argument should be formed by ORing any combination of the following symbols which are defined in fcntl.h:

Value	Meaning
S_IWRITE	Write permission
S_IREAD	Read permission

Since all GEMDOS files are readable, only the S_IWRITE symbol actually has any meaning.

RETURNS

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

SEE

access, chgfa, errno, _OSERR

EXAMPLE

```
/*
 * This piece of code changes file "xyz\pdq.x"
 * so it can be read and written.
 */
#include <fcntl.h>

if(chmod("xyz\pdq.x",S_IWRITE | S_IREAD))
  perror("Change mode");
```

clearerr, clrerr — Clear buffered I/O error flag

Class: ANSI *Category: Stream I/O*

SYNOPSIS

```
#include <stdio.h>

clearerr(fp);
clrerr(fp);

FILE *fp; file pointer
```

DESCRIPTION

The clearerr and clrerr functions clear the error flag associated with the specified file that was previously opened via fopen. Once set, the error flag forces an EOF return any time the file is accessed until the flag is reset.

Note that clearerr is implemented as both a macro and a function. To get the function instead of the macro, include the following line after the #include line:

```
#undef clearerr
```

(The function clrerr is provided for compatibility with some older versions of UNIX.)

SEE

fopen

clock — Determine the processor time used

Class: ANSI *Category: Date and Time*

SYNOPSIS

```
#include <time.h>

time = clock();

clock_t time; clock time since start of execution
```

DESCRIPTION

The clock function determines the processor time used by the process. The clock is started when the process starts and then clock returns the time elapsed since then.

RETURNS

To determine the time in seconds, the value returned by the clock function should be divided by the value of the macro CLK_TCK. If the processor time used is not available or its value cannot be represented, the function returns the value ((clock_t)-1). This will never be the case under GEMDOS.

EXAMPLE

```
/*
 * time a function, returning a value in seconds
 */

#include <time.h>

long time_me(void (*f)(void))
{
  clock_t start;

  start=clock();
  f();
  return (long)((clock()-start)/CLK_TCK);
}
```

close

Close an unbuffered file

Class: UNIX

Category: Low-Level I/O

SYNOPSIS

```
#include <fcntl.h>

error = close(fh);

int error;      non-zero if error
int fh;         file handle
```

DESCRIPTION

This function closes a file that was previously opened via the open function. If there is any pending output, it is completed and the file directory is updated.

All files are automatically closed when your program terminates, but it is good programming practice to close a file when you are finished with it. One reason for doing this is to free up the operating system resources (e.g., control blocks and buffers) that are allocated for the file while it remains open.

RETURNS

The function returns 0 if it is successful. Otherwise, it returns -1 and places additional error information into errno and _OSERR.

SEE

errno, open, _OSERR

EXAMPLE

See the open function.

cmul

Complex product

Class: Lattice

Category: Mathematics

SYNOPSIS

```
#include <math.h>

z = cmul(x,y,z);

struct complex {
  double re;
  double im;
} *z;

struct complex *x, *y;
```

DESCRIPTION

The cmul function calculates the complex product of complex numbers pointed to by x and y, and place the results in the complex number pointed to by z. The pointer z is returned by the function.

For instance, the expression:

```
z = cmul(x,y,z)
```

produces the following assigment:

```
z->re = (x->re * y->re) - (x->im * y->im);
z->im = (x->re * y->im) + (x->im * y->re);
```