Lattice C 5

the C Compiler for your Atari ST Computer

Volume II

Library Manual

Requires:

- ✓ Atari 520ST upwards
  (1M+ memory advised)
- ✓ Disk drive
  (2 floppies or hard disk advised)
- ✓ Mouse

HiSoft
High Quality Software

Lattice C

The C system for your Atari ST

Volume II
Library Manual

Published by HiSoft

Version 5
First edition March 1990 (ISBN 0 948517 30 1)
Second edition April 1991

ISBN for this volume 0 948517 38 7

ISBN for complete 3 volume set 0 948517 28 X

Set using an Apple Macintosh™ and Laserwriter™ with Microsoft Word™ and SuperPaint™.

Table of Contents

1 Introduction

This volume describes the Lattice C library, consisting of the Lattice portable library, the ANSI C library and the UNIX functions available to user programs. Note that this does not include the GEMDOS, BIOS, XBIOS, AES, VDI or Line-A functions which are documented separately in the volume 3.

The next section of this manual covers the header files supplied for use with the functions described in this manual. Many of the headers files are as defined by ANSI, but often contain extensions to provide a more flexible interface. Some of the header files are additions to ANSI and provide access to facilities available on the Atari ST in a more consistent manner than by directly calling the OS. The use of these functions greatly enhances portability to other Lattice C compilers.

The main section provides detailed descriptions of the library functions often with examples. All functions are described in the same basic way, with a synopsis, a description of the function as implemented, the input and output parameters and any side effects of the call, and finally any cross-references to other functions which are related or perform similar functions.

The synopses give a brief summary, listing the header file in which the function is declared, the calling syntax and the types of the parameters.

The calling form is listed as a one line summary, for instance fopen is:

```
#include <stdio.h>

fp = fopen(name, mode);
```

so that the function takes two parameters name and mode returning a single parameter. If the function does not return a value (i.e. 'returns void') then this is indicated by the return value not being assigned.

The types of parameters is then listed; note that the types listed are those used in the *definition*, to call them only compatible types are required. Hence considering fopen, the parameters are:

```
FILE *fp;                   /* function return value in
                               appropriate type */
const char name;            /* first parameter */
const char *mode;           /* second parameter */
```

In general then the types of the parameters you pass would have type (char *) rather than (const char *).

Considering a more complex function such as qsort, the synopsis for this is:

```
#include  <stdlib.h>

qsort(a,n,size,cmp);  Sort a data array

void *a;              data array pointer
size_t  n;            number of elements in array
size_t  size;         element size in bytes
int  (*cmp)(const void *,const void *);
                      pointer to comparison function
```

So that qsort takes four parameters and returns no value. Examining the types of the parameters, the first has type (void *) whose type is compatible with any pointer type (i.e. you may pass any pointer). The second two parameters are of type size_t, hence in general these would passed values of type int. The size_t type was introduced by ANSI and is the type returned by the sizeof operator, (unsigned long) in this implementation. The final parameter is a functional parameter, which takes two pointers to constant objects.

The final form which appears in the synopses are those functions using the ANSI ellipsis operator to indicate a function which takes a variable number of arguments. For instance the printf function synopsis is:

```
#include  <stdio.h>

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

const  char  *fmt;       format  string
```

Hence printf takes a constant format string and a variable number of arguments relating to the formatting string. Note that when using variable argument functions you *must* ensure that you pass an appropriate type as the compiler is unable to check the types of your parameters and promote (or demote) them if necessary.

The fonts used throughout this library manual are:

OCRB	Program fragments and synopses.
Avante Garde	Library identifiers, parameters, disk files and keyboard shortcuts. Note that square brackets (i.e. those used in array accesses) appear as ( ) in this font, whereas parentheses (i.e. those used in function calls) appear as ( ). Beware of the distinction.

Note that *italics* are used solely for emphasis.

2 Header Files

This section describes the header files supplied with the Lattice C compiler, listing the header file and any macros, functions and types declared within them. To gain access to the facilities in these files you must #include them into your program.

For functions declared in a header file, the prototypes are listed so that you can see the types which the parameters should take and the value returned. In general description is not provided on these and you should refer to the main library section for full details.

Any types declared in a header file are listed together with their use. Note that many types were added by the ANSI standardisation committee and so may not be familiar, even to experienced C programmers.

Macros which provide a function like facility are listed in a functional form. Note that in general you may #undef these macros to obtain a true function implementation.

For macros which provide constant values these are indicated as 'const int' which will in general be the type assigned to these expressions. Note that it is important to be aware that these values are expressions and not simple variables as this can lead to unexpected type assignments (e.g. -32768 is the long integer value 32768 negated by the unary minus operator, giving a long integer type to the 'constant').

For external variables made available by a header these are marked as 'extern', and in general you may redefine this yourself to change the default initialiser, or alter them at runtime.

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.

assert.h — Program validation macros

Class: ANSI *Category: Debugging*

SYNOPSIS

```
void assert(int);
```

DESCRIPTION

The assert.h header file contains the definition for the assert macro, which is used to insert diagnostics into a program during debugging, which can then be removed at final compilation time by defining the symbol NDEBUG, causing all references to assert to be removed during the pre-processing phase.

basepage.h — Program basepage definitions

Class: GEMDOS *Category: Process Environment*

SYNOPSIS

```
typedef struct _base
{
    void            *p_lowtpa;      bottom of TPA
    void            *p_hitpa;       top of TPA + 1
    void            *p_tbase;       base of text segment
    long            p_tlen;         length of text
    void            *p_dbase;       base of data segment
    long            p_dlen;         length of data
    void            *p_bbase;       base of BSS segment
    long            p_blen;         length of BSS
    void            *p_dta;         current DTA pointer
    struct _base    *p_parent;      parent's basepage
    void            *p_reserved;
    char            *p_env;         environment strings
    long            p_undef[20];
    char            p_cmdlin[128];  command line image
} BASEPAGE;


extern BASEPAGE *_pbase;  program's basepage pointer
```

DESCRIPTION

The basepage.h header file contains definitions relating to the GEMDOS basepage structure (usually called Program Segment Prefix (PSP) under MS-DOS).

The _pbase variable is used to gain access to the current program's basepage.

Note that this file is included by dos.h.

conio.h — Console I/O declarations

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

SYNOPSIS

```
int  cget(void);
int  cgetc(void);
char *cgets(char *);
int  cputc(int);
int  cputs(const char *);
int  cprintf(const char *, ...);
int  cscanf(const char *, ...);
int  getch(void);
int  getche(void);
int  kbhit(void);
int  iskbhit(void);
int  putch(int);
int  ungetch(int);
```

DESCRIPTION

The conio.h header file contains definitions for console input and output. These functions read and write characters directly at the GEMDOS level. They do not work through any layer of the file manager (i.e. buffered or unbuffered I/O) and so these functions will always return a key immediately one is requested, or write the character as soon as it is sent.

Note that traditionally these functions have been defined in the Lattice dos.h header file, and if you require portability to other Lattice compilers you should include dos.h rather than this file directly (which is included by dos.h anyway).

ctype.h — Character classification and conversion

Class: ANSI *Category: Character Classification/Conversion*

SYNOPSIS

```
int  isalpha(int);      true if c is alpha
int  isupper(int);      true if c is upper case
int  islower(int);      true if c is lower case
int  isdigit(int);      true if c is a digit (0 to 9)
int  isxdigit(int);     true if c is a hexadecimal digit
                        (0 to 9, A to F, a to f)
int  isspace(int);      true if c is white space
int  ispunct(int);      true if c is punctuation
int  isalnum(int);      true if c is alpha or digit
int  isprint(int);      true if c is printable
                        (including blank)
int  isgraph(int);      true if c is graphic (excluding
                        blank)
int  iscntrl(int);      true if c is control character
int  isascii(int);      true if c is ASCII
int  iscsym(int);       true if valid character for C
                        symbols
int  iscsymf(int);      true if valid first character
                        for C symbols
int  _toupper(int);     convert lower case to upper case
int  _tolower(int);     convert upper case to lower case
int  toascii(int);      convert character to ascii
int  toupper(int);      convert character to upper case
int  tolower(int);      convert character to lower case
```

DESCRIPTION

The ctype.h header file contains macros for classifying (is...) and for converting characters (to...).

The is... functions return a non-zero value when the character falls into the category of the function. The to... functions return the character converted as required. Note that the is... functions are normally implemented as macros. If you wish to force the use of an equivalent function the macro should be undefined using #undef is....

Note that the functions isascii, iscsym, iscsymf, _toupper, _tolower, and toascii do not form part of the ANSI C standard.

dirent.h — File system independent directory manipulation

Class: POSIX — *Category: Directory Manipulation*

SYNOPSIS

```
struct dirent
{
    int d_attr;             /* GEMDOS file attribute */
    time_t d_time;          /* time */
    size_t d_size;          /* file size */
    char d_name[FMSIZE];    /* directory entry name */
};

typedef ... DIR;

DIR *opendir(const char *);
struct dirent *readdir(DIR *);
void closedir(DIR *);
void seekdir(DIR *,long );
long telldir(DIR *);
void rewinddir(DIR *);
```

DESCRIPTION

The dirent.h header file contains functions for manipulating directory entries in an OS independent manner. A directory is first opened using the opendir function and entries are then obtained from it using the readdir function. Seeking may also be performed in a manner similar to the buffered I/O subsystem using the seekdir, telldir and rewinddir functions.

The readdir command returns a pointer to the structure shown above, the only element of which you should rely upon being present is the d_name field. The GEMDOS specific entries are obviously not available under other operating systems.

dos.h — OS interface functions and definitions

Class: GEMDOS — *Category: DOS Interface*

SYNOPSIS

```
const int SECSIZ;           disk sector size
const int FNSIZE;           maximum file node size
const int FMSIZE;           maximum file name size
const int FESIZE;           maximum file extension size

extern short _tos;          tos version number
extern short _country;      OS country code
extern long _MSTEP;         OS memory increment
extern long volatile _OSERR;
                            last OS error
extern unsigned long int _STACK;
                            default stack size


struct DISKINFO
{
  unsigned long free; /* number of free clusters */
  unsigned long cpd;  /* clusters per drive */
  unsigned long bps;  /* bytes per sector */
  unsigned long spc;  /* sectors per cluster */
};

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 */
};

long _dclose(int);
long _dcreat(const char *, int);
long _dcreatx(const char *, int);
int _ddup(int);
int _ddup2(int, int);
int _disatty(int);
long _dopen(const char *, int);
long _dread(int,void *, long);
long _dwrite(int,const void *,long);
long _dseek(int, long, int);
int dfind(struct FILEINFO *, const char *, int);
int dnext(struct FILEINFO *);
int getcd(int, char *);
int getfa(const char *);
int chgfa(const char *, int);
int getdsk(void);
int chgdsk(int);
void chgdta(struct FILEINFO *);
struct FILEINFO *getdta(void);
int getdfs(int,struct DISKINFO *);
long getft(int);
int chgft(int, long);
Long ftpack(const char *);
void ftunpk(long, char *);
```

```
int  chgclk(unsigned  char  *);
void  getclk(unsigned  char  *);
int  getpf(char  *,const  char  *);
int  getpfe(char  *,const  char  *);
__stdargs  void  _stub(void);
__stdargs  void  _xcovf(void);
int  onbreak(int  (*)());
int  poserr(const  char  *);
void  geta4(void);
void  __emit(short);
long  getreg(int);
void  putreg(int,  long);
const  int  REG_D0;          register  D0  for  getreg/putreg
const  int  REG_D1;          register  D1  for  getreg/putreg
const  int  REG_D2;          register  D2  for  getreg/putreg
const  int  REG_D3;          register  D3  for  getreg/putreg
const  int  REG_D4;          register  D4  for  getreg/putreg
const  int  REG_D5;          register  D5  for  getreg/putreg
const  int  REG_D6;          register  D6  for  getreg/putreg
const  int  REG_D7;          register  D7  for  getreg/putreg
const  int  REG_A0;          register  A0  for  getreg/putreg
const  int  REG_A1;          register  A1  for  getreg/putreg
const  int  REG_A2;          register  A2  for  getreg/putreg
const  int  REG_A3;          register  A3  for  getreg/putreg
const  int  REG_A4;          register  A4  for  getreg/putreg
const  int  REG_A5;          register  A5  for  getreg/putreg
const  int  REG_A6;          register  A6  for  getreg/putreg
const  int  REG_A7;          register  A7  for  getreg/putreg
```

DESCRIPTION

The dos.h header file contains functions for interfacing with GEMDOS, some of the internal library structures, and OS specific constants.

The major functions supplied by this library are the _d... functions which provide the libraries' interface to GEMDOS, resolving many of the anomalies encountered in manipulating GEMDOS directly.

Assorted file system manipulation functions are also provided together with the facilities to map GEMDOS return values into the standard library structures, via functions such as ftunpk, getclk etc.

This header file also includes the conio.h, basepage.h and osbind.h headers files for convenience.

errno.h

UNIX error definitions

Class: ANSI

Category: Errors

SYNOPSIS

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

DESCRIPTION

The errno.h header file contains the ANSI errno variable which gives details of the last error encountered by the runtime library.

The sys_nerr and sys_errlist items give a count of the errors which may be produced, and a list of the error messages which the values of errno correspond to. Several macros (E...) are provided to give meaningful names to the error numbers produced and these are identical to those produced under UNIX.

Note that the sys_nerr, sys_errlist variable and the E... macros do *not* form part of the ANSI C standard.

fcntl.h — Unbuffered UNIX I/O

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

SYNOPSIS

```
int open(const char *, int, ...);
int opene(const char *, int, int, char *);
long read(int, void *, size_t);
long write(int, const void *, size_t);
int creat(const char *, int);
long lseek(int, long, int);
long tell(int);
int close(int);
int iomode(int, int);
int isatty(int);
long filelength(int);

int rename(const char *,const char *);
int remove(const char *);
int unlink(const char *);

const int O_RDONLY;   open in read only mode
const int O_WRONLY;   open in write only mode
const int O_RDWR;     open in read/write mode
const int O_APPEND;   allow only appends
const int O_CREAT;    creat file if absent
const int O_TRUNC;    truncate file if present
const int O_EXCL;     exclusive create flag
const int O_RAW;      open in untranslated mode

const int S_IREAD;    allow read access
const int S_IWRITE;   allow write access
const int S_IEXEC;    allow execute access
```

DESCRIPTION

The fcntl.h header file contains the interface definitions for the unbuffered I/O sub-system. The open, read, write, creat, lseek, tell, close, iomode, isatty and filelength functions manipulate a file given a library file handle. Note that the handles used by these functions are *not* GEMDOS file handles and you should use the chkufb function (defined in ios1.h) for access to the GEMDOS handle.

Several macros (O_... and S_...) are also defined in this file for use with the creat and open functions.

Note that this header file and its associated defintions do *not* form part of the ANSI C standard.

float.h — Define computational limits for real numbers

Class: ANSI *Category: Mathematics*

SYNOPSIS

```
const int FLT_GUARD;
const int FLT_NORMALIZE;
const int FLT_RADIX;
const int FLT_ROUNDS;

const int DBL_DIG;
const double DBL_EPSILON;
const int DBL_MANT_DIG;
const double DBL_MAX;
const int DBL_MAX_10_EXP;
const int DBL_MAX_EXP;
const double DBL_MIN;
const int DBL_MIN_10_EXP;
const int DBL_MIN_EXP;

const int FLT_DIG;
const float FLT_EPSILON;
const int FLT_MANT_DIG;
const float FLT_MAX;
const int FLT_MAX_10_EXP;
const int FLT_MAX_EXP;
const float FLT_MIN;
const int FLT_MIN_10_EXP;
const int FLT_MIN_EXP;

const int LDBL_DIG;
const long double LDBL_EPSILON;
const int LDBL_MANT_DIG;
const long double LDBL_MAX;
const int LDBL_MAX_10_EXP;
const int LDBL_MAX_EXP;
const long double LDBL_MIN;
const int LDBL_MIN_10_EXP;
const int LDBL_MIN_EXP;

const double HUGE_VAL;
```

DESCRIPTION

The float.h header file contains macros giving the limits placed on the accuracy of floating point calculations. A floating point number is defined by:

$$x = s * b^{e} * \sum_{k=1}^{p} f_k * b^{-k},\ e_{min} \le e \le e_{max}$$

where s represents the sign, b the base of the exponent, e the exponent, p the precision of the mantissa (i.e. the number of digits in base b) and f_k the digits of the mantissa.

The prefixes FLT, DBL and LDBL refer respectively to float, double and long double, and the remaining part of the common expressions signify:

_DIG	The number of *decimal* digits of precision available int the appropriate type.
_EPSILON	The smallest number x such $1.0 + x$ is not equal to 1.0.
_MANT_DIG	Number of digits in the floating point mantissa in base FLT_RADIX (i.e. p in the above expression).
_MIN	The smallest absolute number expressible in the appropriate type.
_MIN_EXP	The smallest integer such that the value of FLT_RADIX raised to its power minus 1 is greater than or equal to _MIN.
_MIN_10_EXP	The smallest integer such that 10 raised to its power minus 1 is greater than or equal to _MIN.
_MAX	The largest number expressible in the appropriate type.
_MAX_EXP	The largest integer such that the value of FLT_RADIX raised to its power minus 1 is less than or equal to _MAX.
_MAX_10_EXP	The largest integer such that 10 raised to its power minus 1 is less than or equal to _MAX.

The remaining definitions are:

FLT_GUARD	Determines whether guard digits are used during multiplication. 0 indicates no, 1 indicates yes.
FLT_NORMALIZE	States whether normalisation is required for floating point quantities. 0 indicates no, 1 indicates yes.
FLT_RADIX	The radix of the exponent in the implementation (i.e. the value of b in the above expression).

FLT_ROUNDS	Type of rounding performed during conversion: -1 Indeterminate. 0 Toward zero (truncation). 1 To nearest. 2 To +∞ (i.e. always up). 3 To -∞ (i.e. always down).

The final definition in float.h is HUGE_VAL, this is normally defined in math.h but is duplicated here for convenience.

Note that the FLT_GUARD and FLT_NORMALIZE macros do *not* form part of the ANSI C standard, also note that ANSI places HUGE_VAL in math.h, not in float.h.

ios1.h — Unbuffered I/O interface file

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

SYNOPSIS

```
const int NUFBS;     default number of UNIX file
                     blocks

extern int _iomode; default unbuffered mode
extern int _nufbs;  number of ubs allocated

struct UFB *chkufb(int);
```

DESCRIPTION

The ios1.h header file contains environment definitions for the unbuffered I/O sub-system. The value NUFBS contains the default number of handles which the libraries make available, this value is normally the same as _nufbs. The _iomode variable allows the default translation mode to be changed, whilst chkufb allows translation of library handles to GEMDOS handles.

Note that this header file and its associated defintions do *not* form part of the ANSI C standard.

limits.h — Integral numerical limits

Class: ANSI *Category: Process Environment*

SYNOPSIS

```
const CHAR_BIT;     bits per char

const CHAR_MAX;     max value for char
const CHAR_MIN;     min value for char
const SCHAR_MAX;    max value for signed char
const SCHAR_MIN;    min value for signed char
const UCHAR_MAX;    max value for unsigned char

const SHRT_MAX;     max value for short int
const SHRT_MIN;     min value for short int
const USHRT_MAX;    max value for unsigned short int

const INT_MAX;      max value for short int
const INT_MIN;      min value for short int
const UINT_MAX;     max value for unsigned short int

const LONG_MAX;     max value for long int
const LONG_MIN;     min value for long int
const ULONG_MAX;    max value for unsigned long int

const MB_LEN_MAX;   maximum bytes in a multibyte
                    character
```

DESCRIPTION

The limits.h header file contains macros defining the integral numerical limits of the program environment. Note that the values of CHAR_MAX and CHAR_MIN are dependent on the -cu flag, whilst INT_MAX and INT_MIN are dependent on the -w compiler flag.

You should be aware that these values are numeric constants and are assigned types according to the normal assignment rules, and hence do *not* neccesarily have the type of the limit they represent.

locale.h — Localisation functions and macros

Class: ANSI *Category: Localisation*

SYNOPSIS

```
const int LC_COLLATE;   collation information
const int LC_CTYPE;     character handling
const int LC_MONETARY;  monetary information
const int LC_NUMERIC;   numeric information
const int LC_TIME;      time information
const int LC_ALL;       all information

struct lconv { ... };   monetary conversion
                        information

extern char DECPT;      local decimal point character

char *setlocale(int, const char *);
struct lconv *localeconv(void);

typedef ... wchar_t;    wide character type
```

DESCRIPTION

The locale.h header file declares functions and macros used for manipulating a program's locale. Note that the ANSI places the type wchar_t in the stddef.h and stdlib.h headers files, and its declaration here is for convenience.

Note that the lconv structure is documented under the localeconv function in the main part of this manual, whilst the LC... macros are discussed under setlocale.

m68881.h — Unary maths co-processor transcendental interface

Class: Lattice *Category: Mathematics*

SYNOPSIS

```
double acos(double);     arc-cosine
double asin(double);     arc-sine
double atan(double);     arc-tangent
double cos(double);      cosine
double cosh(double);     hyperbolic cosine
double exp(double);      exponential
double fabs(double);     absolute value
double fatanh(double);   hyperbolic arc-tangent
double fetoxm1(double);  exponential - 1
double fgetexp(double);  get exponent
double fgetman(double);  get mantissa
double fintrz(double);   integer part, round to zero
double flog2(double);    log base 2
double flognp1(double);  log (n+1)
double fneg(double);     negate
double ftentox(double);  10 to x
double log(double);      log
double log10(double);    log base 10
double pow2(double);     2 to x
double sin(double);      sine
double sinh(double);     hyperbolic sine
double sqrt(double);     square root
double tan(double);      tangent
double tanh(double);     hyperbolic tangent
```

DESCRIPTION

The m68881.h header file declares functions for the standard transcendental functions implemented by the M68881 which take a single argument. Note that the use of these functions requires use of -f8 on lc1, also the code generated *requires* a 68020, 68030 or suitable Line-F emulator to run., Specifically it will *not* work the Atari I/O mapped M68881 card.

Note that this header file and some its defintions do *not* form part of the ANSI C standard.

math.h — Mathematical definitions and declarations

Class: ANSI *Category: Mathematics*

SYNOPSIS

```
const double HUGE_VAL;

double acos(double);
double asin(double);
double atan(double);
double atan2(double, double);
double ceil(double);
double cos(double);
double cosh(double);
double exp(double);
double fabs(double);
double floor(double);
double fmod(double, double);
double frexp(double, int *);
double ldexp(double, int);
double log(double);
double log10(double);
double modf(double, double *);
double pow(double, double);
double sin(double);
double sinh(double);
double sqrt(double);
double tan(double);

struct exception
{
   int     type;          error type
   char    *name;         math function name
   double  arg1, arg2;    function arguments
   double  retval;        proposed return value
};

const int DOMAIN;      domain error
const int OVERFLOW;    overflow
const int PLOSS;       partial loss of significance
const int RANGE;       range error
const int SING;        singularity
const int TLOSS;       total loss of significance
const int UNDERFLOW;   underflow

const int FPECOM;      not comparable
const int FPENAN;      not a number
const int FPEOVF;      overflow
const int FPEUND;      underflow
const int FPEZDV;      zero divisor

extern int _FPERR;     low-level floating point error
                       status

const double PI;       PI
const double PID2;     PI/2
const double PID4;     PI/4
const double I_PI;     1/PI
const double I_PID2;   1/(PI/2)
```



```
const double HUGE;     largest representable absolute
                       double
const double TINY;     smallest representable absolute
                       double
const double LOGHUGE;  ln(HUGE);
const double LOGTINY;  ln(TINY);

double cot(double);
double drand48(void);
double erand48(unsigned short *);
double except(int, char *, double, double, double);
char *ecvt(double, int, int *, int *);
char *fcvt(double, int, int *, int *);
char *gcvt(double, int, char *);
long jrand48(unsigned short *);
void lcong48(unsigned short *);
long lrand48(void);
int matherr(struct exception *);
long mrand48(void);
long nrand48(unsigned short *);
double pow2(double);
unsigned short *seed48(unsigned short *);
void srand48(long);
double tanh(double);
```

DESCRIPTION

The math.h header file declares functions and macros for the mathematical functions.

Note that some of the defintions in this header do *not* form part of the ANSI C standard.

oserr.h — TOS error definitions

Class: GEMDOS *Category: Errors*

SYNOPSIS

```
extern int volatile _OSERR;   TOS error number
extern int os_nerr;           number of error codes
extern char *os_errlist[];    TOS error messages
const int E...;               error names
```

DESCRIPTION

The oserr.h header file contains the operating system error variable _OSERR, which gives details of the last OS error encountered by the runtime library.

The os_nerr and os_errlist items give a count of the errors which may be produced, and a list of the error messages which the values of _OSERR correspond to. Several macros (E...) are provided to give meaningful names to the error numbers produced.

Note that this header file and its associated definitions do *not* form part of the ANSI C standard.

setjmp.h — Declarations for non-local jumps

Class: ANSI *Category: Non-Local Jumps/Signal Handling*

SYNOPSIS

```
typedef ... jmp_buf;          jump buffer type

int setjmp(jmp_buf);
void longjmp(jmp_buf,int);
```

DESCRIPTION

The setjmp.h header file contains the declarations for non-local jumps. You should be aware of the potential problems using these functions, discussed under the main setjmp entry in this manual.

signal.h — Signal handling routines

Class: ANSI *Category: Non-Local Jumps/Signal Handling*

SYNOPSIS

```
const int SIGABRT;     abnormal termination, abort()
const int SIGFPE;      floating point exception
const int SIGILL;      illegal instruction
const int SIGINT;      interrupt from GEMDOS
const int SIGSEGV;     segmentation violation
const int SIGTERM;     termination request

void  (*SIG_DFL)(int); default action
void  (*SIG_IGN)(int); ignore the signal
void  (*SIG_ERR)(int); error return

void  (*signal(int,void  (*)(int)))(int);
int  raise(int);

typedef ... sig_atomic_t; signal atomic type
```

DESCRIPTION

The signal.h header file contains the definitions and declarations for signal handling. Note that the signals provided are those required by ANSI however these are not necessarily called at any other time than explicitly via raise.

The type sig_atomic_t is a type which is guaranteed to be accessed atomically if simultaneous signals occur, however any variable definition *must* include the volatile modifier viz:

```
volatile  sig_atomic_t  sig_count;
```


stdarg.h — ANSI variable argument header

Class: ANSI *Category: Variable Argument Handling*

SYNOPSIS

```
typedef ... va_list; variable list type

void  va_start(va_list,typename);
typename  *va_arg(va_list,typename);
void  va_end(va_list)
```

DESCRIPTION

The stdarg.h header file contains routines for manipulating variable numbers of arguments in an ANSI fashion. Note that the header file varargs.h provides similar facilities (and under similar names), but follows the UNIX definition.

EXAMPLE

```
/*
 * concatenate a variable number of strings,
 * terminated by NULL into a malloced block
 * of memory
 */

#include  <stdarg.h>
#include  <string.h>
#include  <stdlib.h>

char *strcatl(const  char  *s1, ...)
{
  va_list  strings;
  size_t  length;
  char  *s, *concat;

  va_start(strings, s1);
  length=strlen(s1);          /* fetch length of first
                                 string */
  while  (s=va_arg(strings,  char  *))
    length+=strlen(s);        /* add in remaining string
                                 lengths */
  va_end(strings);            /* all done this pass */

  if  (concat=malloc(length+1)) /* get some RAM */
  {
    va_start(strings,  s1);
    strcpy(concat,  s1);      /* copy first string */
    while  (s=va_arg(strings,  char  *))
      strcat(concat,  s);     /* concatenate rest */
    va_end(strings);
  }
  return  concat; /* return composite string */
}
```

stddef.h — ANSI standard definitions

Class: ANSI — *Category: Process Environment*

SYNOPSIS

```
typedef  ...  size_t;        type of sizeof
typedef  ...  ptrdiff_t;     type of pointer difference
typedef  ...  wchar_t;       wide character type

size_t  offsetof(type,memb); obtain field offset

void  *NULL;                 NULL pointer constant
```

DESCRIPTION

The stddef.h header file contains ANSI definitions for the types of compiler and library quantities.

The offsetof macro may be used for obtaining the byte offset of a field within an aggregate item.

stdio.h — Standard I/O library definitions

Class: ANSI — *Category: Stream I/O*

SYNOPSIS

```
typedef  ...  FILE;      FILE type
typedef  ...  fpos_t;    file position type

const  int  FILENAME_MAX;    max chars in a filename
const  int  FOPEN_MAX;       max number of open files

const  int  _IOFBF;      fully buffered flag
const  int  _IONBF;      non-buffered flag
const  int  _IOLBF;      line-buffered flag

const  int  BUFSIZ;      standard buffer size
const  int  EOF;         end-of-file code
const  int  L_tmpnam;    maximum tmpnam filename length

const  int  SEEK_SET;    seek from beginning of file
const  int  SEEK_CUR;    seek from current file position
const  int  SEEK_END;    seek from end of file

const  int  TMP_MAX;     maximum unique temporary files

FILE  *stdin;            standard input file pointer
FILE  *stdout;           standard output file pointer
FILE  *stderr;           standard error file pointer
FILE  *stdaux;           standard auxiliary file pointer
FILE  *stdprt;           standard printer file pointer

int  rename(const char *,const char *);
int  remove(const char *);
FILE  *tmpfile(void);
char  *tmpnam(char *s);
int  fclose(FILE *);
int  fflush(FILE *);
FILE *fopen(const char *, const char *);
FILE *freopen(const char *, const char *, FILE *);
void  setbuf(FILE *, char *);
int  setvbuf(FILE *, char *, int, size_t);
int  fprintf(FILE *, const char *, ...);
int  fscanf(FILE *, const char *, ...);
int  printf(const char *, ...);
int  scanf(const char *, ...);
int  sprintf(char *, const char *, ...);
int  sscanf(const char *, const char *, ...);
int  vfprintf(FILE *, const char *, va_list);
int  vprintf(const char *, va_list);
int  vsprintf(char *, const char *, va_list);

int  fgetc(FILE *);
char  *fgets(char *, int, FILE *);
int  fputc(int, FILE *);
int  fputs(const char *, FILE *);
int  getc(FILE *);
int  getchar(void);
char  *gets(char *);
int  putc(int, FILE *);
```

```
int  putchar(int);
int  puts(const  char  *);
int  ungetc(int,  FILE  *);
size_t  fread(void  *,  size_t,  size_t,  FILE  *);
size_t  fwrite(const  void  *,  size_t,  size_t,  FILE  *);
int  fgetpos(FILE  *,  fpos_t  *);
int  fseek(FILE  *,  long  int,  int);
int  fsetpos(FILE  *,  const  fpos_t  *);
long  int  ftell(FILE  *);
void  rewind(FILE  *);
void  clearerr(FILE  *);
int  feof(FILE  *);
int  ferror(FILE  *);
void  perror(const  char  *);

int  fcloseall(void);
FILE  *fdopen(int,  const  char  *);
int  fgetchar(void);
int  fileno(FILE  *);
int  flushall(void);
void  fmode(FILE  *,  int);
int  fputchar(int);
int  setnbf(FILE  *);
int  access(const  char  *,  int);
int  chdir(const  char  *);
int  chmod(const  char  *,  int);
char  *getcwd(char  *,  int);
int  mkdir(const  char  *);
int  rmdir(const  char  *);
FILE  *fopene(const  char  *,  const  char  *,char  *);
int  unlink(const  char  *);

char  *mktemp(char  *s);
short  fputw(short,FILE  *);
long  fputl(long,FILE  *);
short  fgetw(FILE  *);
long  fgetl(FILE  *);

extern  unsigned  long  __fmask;  default  file  mask
extern  int  _fmode;    default  access  mode
extern  int  _bufsiz;   default  file  buffer  size
```

DESCRIPTION

The stdio.h header file contains definitions, declarations and macros for use by the ANSI standard input/output library.

The following functions and variables which appear in this header do not form part of the ANSI C standard: __fmask, _bufsiz, _fmode, access, chdir, chmod, fcloseall, fdopen, fgetchar, fgetl, fgetw, fileno, flushall, fmode, fopene, fputchar, fputl, fputw, getcwd, mkdir, mktemp, rmdir, setnbf and unlink.

stdlib.h

Standard utility definitions

Class: ANSI

Category: General Functions

SYNOPSIS

```
extern  char  MB_CUR_MAX;

typedef  ...  div_t;          div()  type

typedef  ...  ldiv_t;         ldiv()  type

void  *malloc(size_t);
void  *calloc(size_t,size_t);
void  *realloc(void  *,  size_t);
void  free(void  *);

void  *getml(size_t);
int  rlsml(void  *,  size_t);
size_t  sizmem(void);
size_t  chkml(void);

void  *getmem(unsigned);
int  rlsmem(void  *,  unsigned);

void  *alloca(size_t);

extern  size_t  _stkdelta;  stack/data  chicken  factor

void  *sbrk(unsigned);
void  *lsbrk(long);

int  chdir(const  char  *);
int  chmod(const  char  *,  int);
char  *getcwd(char  *,  int);
int  mkdir(const  char  *);
int  rmdir(const  char  *);

void  qsort(void  *,  size_t,  size_t,
                int  (*)(const  void  *,  const  void  *));
void  dqsort(double  *,size_t);
void  fqsort(float  *,size_t);
void  lqsort(long  *,size_t);
void  sqsort(short  *,size_t);
void  tqsort(char  **,size_t);

void  bsearch(const  void  *,  const  void  *,  size_t,
          size_t,  int  (*)(const  void  *,  const  void  *));

int  mblen(const  char  *,size_t);
size_t  mbstowcs(wchar_t  *,  const  char  *,  size_t);
int  mbtowc(wchar_t  *,  const  char  *,  size_t);
size_t  wcstombs(char  *,  const  wchar_t  *,  size_t);
int  wctomb(char  *,  wchar_t);
```

```
void  exit(int);
void  abort(void);
int atoi(const char *);
double atof(const char *);
long int atol(const char *);
char *getenv(const char *);

void  _exit(int);
void  _XCEXIT(int);
char *argopt(int, char *[], char *, int *, char *);

void *lsearch(const void *, void *, size_t *, size_t,
              int (*)(const void *, const void *));
void *lfind(const void *, const void *,
            const size_t *, size_t,
            int (*)(const void *, const void *));

int  getpid(void);
int  getopt(int argc, const char *argv[],
                      const char *optstring);

extern int optopt,opterr,optind;
extern char *optarg;

int system(const char *);
size_t _hash(const char *);

int  abs(int);

long atol(char *);
char *ecvt(double, int, int *, int *);
char *fcvt(double, int, int *, int *);
char *gcvt(double, int, char *);
long getfnl(const char *, char *, size_t, int);
int  iabs(int);
long  labs(long);
int  onexit(int(*)(int));
int  putenv(char *);

int  rand(void);
int  rmvenv(const char *);
void srand(unsigned int);
double strtod(const char *,const char **);
long int strtol(const char *, char **, int);
unsigned long int strtoul(const char *,char **,int);
long int utpack(const char *);
void utunpk(long int, char *);

int atexit(void (* )(void));
div_t div(int, int);
ldiv_t ldiv(long int, long int);

unsigned long _lrotl(unsigned long,int);
unsigned short _rotl(unsigned short,int);
unsigned long _lrotr(unsigned long,int);
unsigned short _rotr(unsigned short,int);
```



```
int  forkl(const char *,...);
int  forkle(const char *,...);
int  forklp(const char *,...);
int  forklpe(const char *,...);
int  forkv(const char *,const char **);
int  forkve(const char *,const char **,const char **);
int  forkvp(const char *,const char **);
int  forkvpe(const char *,const char **,
                                const char **);

int  wait(void);

const int EXIT_SUCCESS;   success exit value
const int EXIT_FAILURE;   failure exit value;

const int RAND_MAX;   maximum rand() value
```

DESCRIPTION

The stdlib.h header file contains general utility definitions, declarations and macros defined by the ANSI standard.

The following functions and variables which appear in this header do not form part of the ANSI C standard: _exit, _hash, _lrotl, _lrotr, _rotl, _rotr, _stkdelta, _XCEXIT, alloca, argopt, chdir, chkml, chmod, dqsort, ecvt, fcvt, forkl, forkle, forklp, forklpe, forkv, forkve, forkv, forkvpe, fqsort, gcvt, getcwd, getfnl, getmem, getml, getopt, getpid, labs, lfind, lqsort, lsbrk, lsearch, mkdir, onexit, optarg, opterr, optind, optopt, putenv, rlsmem, rlsml, rmdir, rmvenv, sbrk, sizmem, sqsort, tqsort, utpack, utunpk and wait.

string.h — String manipulation

Class: ANSI

Category: String manipulation

SYNOPSIS

```
extern char _SLASH;   path separator character

char *strcat(char *, const char *);
char *strchr(const char *, int);
int strcmp(const char *, const char *);
char *strcpy(char *, const char *);
size_t strcspn(const char *, const char *);
size_t strspn(const char *, const char *);
size_t strlen(const char *);
char *strncat(char *, const char *, size_t);
int strncmp(const char *, const char *, size_t);
char *strncpy(char *, const char *, size_t);
char *strpbrk(const char *, const char *);
char *strrchr(const char *, int);
char *strstr(const char *, const char *);
char *strtok(char *, const char *);
char *strerror(int);
int strcoll(const char *, const char *);
size_t strxfrm(char *, const char *, size_t);

size_t stcarg(const char *, const char *);
size_t stccpy(char *, const char *, size_t);
char *stpcpy(char *, const char *);
char *strdup(const char *);
void strins(char *, const char *);
char *strnset(char *, int, size_t);
char *strrev(char *);
size_t stcis(const char *, const char *);
size_t stcisn(const char *, const char *);
size_t stcpm(const char *, const char *, char **);
size_t stcpma(const char *, const char *);
char *stpblk(const char *);
char *stpbrk(const char *, const char *);
char *stpchr(const char *, int);
char *stpsym(const char *, char *, size_t);
char *stpchrn(const char *, int);
char *stptok(const char *, char *, size_t,
                                   const char *);
long strbpl(char **, size_t, const char *);

int stcd_i(const char *, int *);
int stcd_l(const char *, long *);
int stch_i(const char *, int *);
int stch_l(const char *, long *);
int stci_d(const char *, int);
int stci_h(const char *, int);
int stci_o(const char *, int);
int stcl_d(const char *, long);
int stcl_h(const char *, long);
int stcl_o(const char *, long);
int stco_i(const char *, int *);
int stco_l(const char *, long *);
```



```
int stcgfe(char *, char *);
int stcgfn(char *, char *);
int stcgfp(char *, const char *);

int stcsma(char *, char *);
int stcu_d(char *, unsigned);
int stcul_d(char *, unsigned long);
size_t stclen(const char *);
char *stpdate(char *, int, char *);
char *stptime(char *, int, char *);
int strmid(const char *, char *, size_t, size_t);
char *strlwr(char *);
void strmfe(char *, const char *, const char *);
void strmfn(char *, const char *, const char *,
                        const char *, const char *);
void strmfp(char *, const char *, const char *);
int strnicmp(const char *, const char *, size_t);
int stricmp(const char *, const char *);

char *strset(char *, int);
void strsfn(const char *, char *, char *, char *,
                                                char *);

char *strupr(char *);
int stspfp(char *, int *);
void strsrt(char *[], size_t);

void *memchr(const void *, int, size_t);
int memcmp(const void *, const void *, size_t);
void *memcpy(void *, const void *, size_t);
void *memmove(void *, const void *, size_t);
void *memset(void *, int, size_t);
void *memccpy(void *, const void *, int, size_t);
void *memswp(void *, void *, size_t);
void *memrep(void *, void *, size_t, size_t);
void setmem(void *, unsigned, int);
void movmem(void *, void *, unsigned);
void repmem(void *, void *, unsigned, unsigned);
void swmem(void *, void *, unsigned);
```

DESCRIPTION

The string.h header file contains the definitions for handling strings and buffers via the standard library.

This header file contains many functions which do not form part of the ANSI C standard, the functions which *do* appear therein are: memchr, memcmp, memcpy, memmove, memset, strcat, strchr, strcmp, strcoll, strcpy, strcspn, strerror, strlen, strncat, strncmp, strncpy, strpbrk, strrchr, strspn, strstr, strtok and strxfrm.