Introl−CODE Runtime Libraries

Introl Corporation

Table of Contents

Table of Contents

Table of Contents

Introl−CODE Runtime Libraries

This document describes the runtime support libraries that are included with CODE. Two sets of libraries are supplied: The Assembly libraries and the C libraries. We supply the source code for all library routines. We also supply the Assembly libraries pre−assembled for all targets and the C libraries pre−compiled for those targets that are supported by Introl−C.

The Assembly Libraries contain routines that can be called from assembly language and may also contain support routines that the C compiler uses. Both of these libraries, at a minimum, should be linked with each of your C programs. Neither library is required for assembly language programming but you may find some of the pre−written routines useful.

The Assembly Libraries
: The assembler library routines provided with CODE.

The C Libraries
: ANSI and additional library routines provided with Introl−C.

The Assembly Libraries

The routines in this library may be used by assembly language programmers and are used by the C compiler libraries to tailor them to the runtime environment. You may want to review the information on the runtime environment.

68HC05
: The library routines specific to the 68HC05.

68HC08
: The library routines specific to the 68HC08.

6809
: The library routines specific to the 6809.

68HC11
: The library routines specific to the 68HC11, 6301 and 6801.

68HC12
: The library routines specific to the 68HC12.

68HC16
: The library routines specific to the 68HC16.

68XXX
: The library routines specific to the 68XXX family.

68HC05

A pre−assembled support library has been provided with Introl−CODE. The object code for the support library is supplied in the file $INTROL/lib/libgen.a05.

bssinit.s
: Initialize the BSS (uninitialized RAM) area to zero.

config.lib
: Nothing yet.

debugio.s
: Input/Output stubs for debugging.

exithalt.s
: Exit program and stop.

exitreturn.s
: Exit program and return.

hc05c4.ddf
: Device definition file for the the MC68HC05C4.

interruptinit.s
: Enable processor interrupts.

Makefile
: Build the gen library.

n05.ld
: The example linker command file.

nstart.s
: The program startup code.

registers.mac
: A macro set used to build device definition files.

SCI.s
: The default stdio input/output routines. These functions use the on chip SCI port.

stackinit.s
: Initialize the processor's stack pointer.

vecalone.s
: The stand−alone revector table.

vecinit.s
: Initialize the revector table.

vectors.s
: The processor exception vector definitions.

68HC08

A pre−assembled support library has been provided with Introl−CODE. The object code for the support library is supplied in the file $INTROL/lib/libgen.a08.

bssinit.s
: Initialize the BSS (uninitialized RAM) area to zero.

config.lib
: Nothing yet.

debugio.s
: Input/Output stubs for debugging.

exithalt.s
: Exit program and stop.

exitreturn.s
: Exit program and return.

hc708xl36.ddf
: Device definition file for the MC68HC08XL36.

interruptinit.s
: Enable processor interrupts.

Makefile
: Build the gen library.

n08.ld
: The example linker command file.

nstart.s
: The program startup code.

registers.mac
: A macro set used to build device definition files.

SCI.s
: The default stdio input/output routines. These functions use the on chip SCI port.

stackinit.s
: Initialize the processor's stack pointer.

vecalone.s
: The stand−alone revector table.

vecinit.s
: Initialize the revector table.

vectors.s
: The processor exception vector definitions.

6809

A pre−assembled support library has been provided with Introl−CODE. The object code for the support library is supplied in the file $INTROL/lib/libgen.a09.

bssinit.s
: Initialize the BSS (uninitialized RAM) area to zero.

config.lib
: Nothing yet.

crex.lib
: Processor specific CREX realtime executive definitions.

crex.s
: Implements the core of CREX.

CXACIA.s
: Interrupt driven serial stdio input/output routines that work under CREX.

crexstub.s
: Stub routines that can replace CREX for modules that can run without full CREX.

cxalloc.s
: The CREX dynamic memory management module.

cxbqueue.s
: The byte queue module of CREX.

cxclock.s
: The CREX clock module.

cxidle.s
: The default CREX idle thread.

cxmillitimer.s
: The default CREX millisecond calculator for cxclock.s.

cxqueue.s
: The arbitrary element size queue module of CREX.

datainit.s
: Copy the initialized, writable data from ROM to RAM.

debugio.s
: Input/Output stubs for debugging.

exithalt.s
: Exit program and stop.

exitreturn.s
: Exit program and return.

interruptinit.s
: Enable processor interrupts.

Makefile
: Build the gen library.

mc6809.ddf
: Device definition file for the MC6809.

n09.ld
: The example linker command file.

nstart.s
: The program startup code.

ACIA.s
: The default stdio input/output routines. These functions use the an ACIA port.

stackinit.s

Initialize the processor's stack pointer.

vecalone.s

The stand−alone revector table.

vecinit.s

Initialize the revector table.

vectors.s

The processor exception vector definitions.

68HC11

Several pre−assembled versions of the support library have been provided with Introl−CODE. Their file names, descriptions, and the assembler command line required to reassemble the components are listed in the table below. The object code for the support library is supplied in the files $INTROL/lib/libgen.a11, libgen.a01, libgen.a03.

File name	Support library for:	Assembler command line
libgen.a11	68HC11	as11 *filename.s*
libgen.a01	6801/03	as11 −p0=2 *filename.s*
libgen.a03	6301/03	as68 −p0=3 *filename.s*

ACIA.s
: Alternate stdio input/output routines. These functions use a 6850 ACIA port.

BUFFALO.s
: Alternate stdio input/output routines. These functions use the BUFFALO monitor's input/output routines.

bssinit.s
: Initialize the BSS (uninitialized RAM) area to zero.

callrtc.s
: Runtime support for bank switching.

config.lib
: Set the processor type from the −p0 option.

crex.lib
: Processor specific CREX realtime executive definitions.

crex.s
: Implements the core of CREX.

CXSCI.s
: Interrupt driven serial stdio input/output routines that work under CREX.

crexstub.s
: Stub routines that can replace CREX for modules that can run without full CREX.

cxalloc.s
: The CREX dynamic memory management module.

cxalloclist.s
: Allocate a linked list.

cxbqueue.s
: The byte queue module of CREX.

cxclock.s
: The CREX clock module.

cxidle.s
: The default CREX idle thread.

cxmillitimer.s
: The default CREX millisecond calculator for cxclock.s.

cxqueue.s
: The arbitrary element size queue module of CREX.

cxtimeout.s
: The timeout module of CREX.

datainit.s

Copy the initialized, writable data from ROM to RAM.

debugio.s
Input/Output stubs for debugging.

evbsci.s
Set up the EVB board SCI source.

exithalt.s
Exit program and stop.

exitreturn.s
Exit program and return.

fastinit.s
Do chip initializations that have to be done soon after reset.

hc11a8.ddf
Device definition file for the MC68HC11A0, MC68HC11A1, MC68HC11A7, MC68HC11A8.

hc11d3.ddf
Device definition file for the MC68HC11D3.

hc11e.ddf
Device definition file for the MC68HC11E0, MC68HC11E1, MC68HC11E2, and MC68HC11E9.

hc11f1.ddf
Device definition file for the MC68HC11F1.

hc11k.ddf
Device definition file for the MC68HC11K0, MC68HC11K1, MC68HC11K3, and MC68HC11K4.

hc11p2.ddf
Device definition file for the MC68HC11P2.

hd6303y.ddf
Device definition file for the HD6303Y.

interruptinit.s
Enable processor interrupts.

Makefile
Build the gen library.

mc6801.ddf
Device definition file for the MC6801.

n11.ld
The example linker command file.

nstart.s
The program startup code.

registers.mac
A macro set used to build device definition files.

SCI.s
The default stdio input/output routines. These functions use the on chip SCI port.

stackinit.s
Initialize the processor's stack pointer.

vecalone.s
The stand−alone revector table.

vecbuffalo.s
Use BUFFALO's revector table.

vecinit.s
Initialize the revector table.

vectors.s
The processor exception vector definitions.

68HC12

A pre−assembled support library has been provided with Introl−CODE. The object code for the support library is supplied in the file $INTROL/lib/libgen.a12.

bssinit.s
: Initialize the BSS (uninitialized RAM) area to zero.

config.lib
: Nothing yet.

crex.lib
: Processor specific CREX realtime executive definitions.

crex.s
: Implements the core of CREX.

CXSCI0.s
: Interrupt driven serial stdio input/output routines that work under CREX.

crexstub.s
: Stub routines that can replace CREX for modules that can run without full CREX.

cxalloc.s
: The CREX dynamic memory management module.

cxalloclist.s
: Allocate a linked list.

cxbqueue.s
: The byte queue module of CREX.

cxclock.s
: The CREX clock module.

cxidle.s
: The default CREX idle thread.

cxmillitimer.s
: The default CREX millisecond calculator for cxclock.s.

cxqueue.s
: The arbitrary element size queue module of CREX.

cxtimeout.s
: The timeout module of CREX.

datainit.s
: Copy the initialized, writable data from ROM to RAM.

debugio.s
: Input/Output stubs for debugging.

exithalt.s
: Exit program and stop.

exitreturn.s
: Exit program and return.

fastinit.s
: Do chip initializations that have to be done soon after reset.

hc12.ddf
: Device definition file for the MC68HC812A4.

hc912b32.ddf
: Device definition file for the MC68HC912B32.

interruptinit.s
: Enable processor interrupts.

Makefile

Build the gen library.

n12.ld
: The example linker command file.

nstart.s
: The program startup code.

registers.mac
: A macro set used to build device definition files.

SCI0.s
: The default stdio input/output routines. These functions use the on chip SCI0 port.

stackinit.s
: Initialize the processor's stack pointer.

vecalone.s
: The stand−alone revector table.

vecinit.s
: Initialize the revector table.

vectors.s
: The processor exception vector definitions.

68HC16

A pre−assembled support library has been provided with Introl−CODE. The object code for the support library is supplied in the file $INTROL/lib/libgen.a16.

bssinit.s
: Initialize the BSS (uninitialized RAM) area to zero.

config.lib
: Nothing yet.

crex.lib
: Processor specific CREX realtime executive definitions.

crex.s
: Implements the core of CREX.

CXSCI.s
: Interrupt driven serial stdio input/output routines that work under CREX.

crexstub.s
: Stub routines that can replace CREX for modules that can run without full CREX.

cxalloc.s
: The CREX dynamic memory management module.

cxbqueue.s
: The byte queue module of CREX.

cxclock.s
: The CREX clock module.

cxidle.s
: The default CREX idle thread.

cxmillitimer.s
: The default CREX millisecond calculator for cxclock.s.

cxqueue.s
: The arbitrary element size queue module of CREX.

datainit.s
: Copy the initialized, writable data from ROM to RAM.

debugio.s
: Input/Output stubs for debugging.

exithalt.s
: Exit program and stop.

exitreturn.s
: Exit program and return.

fbssinit.s
: Initialize the far BSS area to zero.

fdatainit.s
: Copy the initialized, writable far data from ROM to RAM.

hc16y1.ddf
: Device definition file for the MC68HC16Y1.

hc16y3.ddf
: Device definition file for the MC68HC16Y3.

hc16z1.ddf
: Device definition file for the MC68HC16Z1.

hc916x1.ddf
: Device definition file for the MC68HC916X1.

interruptinit.s

Enable processor interrupts.

load_k.s
: Load the K registers with the near bank page prior to starting the user program.

Makefile
: Build the gen library.

n16.ld
: The example linker command file.

nstart.s
: The program startup code.

registers.mac
: A macro set used to build device definition files.

SCI.s
: The default stdio input/output routines. These functions use the on chip SCI port.

siminit.s
: Initialize the SIM module.

sraminit.s
: Initialize the SRAM module.

vecalone.s
: The stand−alone revector table.

vecinit.s
: Initialize the revector table.

vectors.s
: The processor exception vector definitions.

68XXX

Several pre−assembled versions of the support library have been provided with Introl−CODE. Their file names, descriptions, and the assembler command line required to reassemble the components are listed in the table below. The object code for the support library is supplied in the files $INTROL/lib/libgen.a68, libgen.a00, libgen.a10, libgen.a20, libgen.a40, libgenm.a20, and libgenm.a40.

File name	Support library for:	Assembler command line
libgen.a68	683XX	as68 *filename.s*
libgen.a00	68000	as68 −p0=1 *filename.s*
libgen.a10	68010	as68 −p0=2 *filename.s*
libgen.a20	68020/68030 with software floating−point	as68 −p0=3 *filename.s*
libgenm.a20	68020/68030 with 68881/2 floating−point	as68 −p0=3 −p1=1 *filename.s*
libgen.a40	68040 with software floating−point	as68 −p0=7 *filename.s*
libgenm.a40	68040 with hardware floating−point	as68 −p0=7 −p1=2 *filename.s*

bssinit.s
: Initialize the BSS (uninitialized RAM) area to zero.

crex.lib
: Processor specific CREX realtime executive definitions.

crex.s
: Implements the core of CREX.

CXSCI.s
: Interrupt driven serial stdio input/output routines that work under CREX.

crexstub.s
: Stub routines that can replace CREX for modules that can run without full CREX.

cxalloc.s
: The CREX dynamic memory management module.

cxbqueue.s
: The byte queue module of CREX.

cxclock.s
: The CREX clock module.

cxidle.s
: The default CREX idle thread.

cxmillitimer.s
: The default CREX millisecond calculator for cxclock.s.

cxqueue.s
: The arbitrary element size queue module of CREX.

datainit.s
: Copy the initialized, writable data from ROM to RAM.

debugio.s
: Input/Output stubs for debugging.

exithalt.s
: Exit program and stop.

exitreturn.s
: Exit program and return.

interruptinit.s
: Enable processor interrupts.

Makefile
: Build the gen library.

mc68331.ddf
: Device definition file for the MC68331.

mc68332.ddf
: Device definition file for the MC68332.

n68.ld
: The example linker command file.

nstart.s
: The program startup code.

registers.mac
: A macro set used to build device definition files.

SCI.s
: The default stdio input/output routines. These functions use the on chip SCI port.

siminit.s
: Initialize the SIM module.

sraminit.s
: Initialize the SRAM module.

vecalone.s
: The stand−alone revector table.

vecinit.s
: Initialize the revector table.

vectors.s
: The processor exception vector definitions.

The C Libraries

The ANSI−C library was designed to conform to the ANSI standard as it applies to embedded systems, with as little overhead as possible. You may not desire some of the features that have been added or may wish to add your own. You may want to review the information on Configuring the runtime environment.

ANSI−C Library
: The ANSI−C library routines supported by Introl−C.

Introl−C Library
: Addition library rountines provided by Introl−C. This is where you should do C library customization, such as additional stdio streams. This also describes the C interfaces to the CREX realtime executive.

68HC05
: The library routines specific to the 68HC05.

68HC08
: The library routines specific to the 68HC08.

6809
: The library routines specific to the 6809.

68HC11
: The library routines specific to the 68HC11, 6301 and 6801.

68HC12
: The library routines specific to the 68HC12.

68HC16
: The library routines specific to the 68HC16.

68XXX
: The library routines specific to the 68XXX family.

ANSI−C Functions

assert	Diagnostics	ctype	Character handling	errno	Errors
locale	Localization	math	Mathematics	setjmp	Non−local jumps
signal	Signal handling	stdarg	Variable arguments	stddef	Common definitions
stdio	Input/output	stdlib	General utilities	string	String handling
time	Date and time				

A B C DEFGHIJKLMNOPQRSTUVWXYZ

abort
: cause abnormal program termination

abs
: integer absolute value function

acos
: arc cosine function

asctime
: convert date and time to ASCII

asin
: arc sine function

assert
: expression verification macro

atan
: arc tangent function of one variable

atan2
: arc tangent function of two variables

atexit
: register a function to be called on exit

atof
: convert ASCII string to double

atoi
: convert ASCII string to integer

atol
: convert ASCII string to long integer

bsearch
: binary search of a sorted table

calloc
: allocate clean memory (zero initialized space)

ceil
: round to smallest integral value greater than or equal

clearerr
: reset stream status

clock
: determine processor time used

cos
: cosine function

cosh
: hyperbolic cosine function

ctime

convert date and time to ASCII

ctype
character classification macros

difftime
time difference

div
return quotient and remainder from division

exit
perform normal program termination

exp
calculate exponential

fabs
floating−point absolute value function

fclose
close a stream

feof
check a stream for end−of−file

ferror
check a stream for errors

fflush
flush a stream

fgetc
get next character from input stream

fgetpos
get stream position

fgets
get a line from a stream

floor
round to largest integral value not greater than x

fmod
floating−point remainder function

fopen
stream open function

fprintf
formatted output conversion

fputc
output a character to a stream

fputs
output a line to a stream

fread
binary stream input

free
free up memory allocated with malloc, calloc or realloc

freopen
stream reopen function

frexp
convert floating−point number to fractional and integral

fscanf
input format conversion

fseek
reposition a stream

fsetpos
: reposition a stream

ftell
: get a stream position

fwrite
: binary stream output

getc
: get next character from input stream

getchar
: get next character from input stream

getenv
: get environment variable

gets
: get a line from a stream

gmtime
: convert date and time to ASCII

isalnum
: alphanumeric character test

isalpha
: alphabetic character test

isascii
: test for ASCII character

iscntrl
: control character test

isdigit
: decimal−digit character test

isgraph
: printing character test (space character exculsive)

islower
: lower−case character test

isprint
: printing character test (space character inclusive)

ispunct
: punctuation character test

isspace
: white−space character test

isupper
: upper−case character test

isxdigit
: hexadecimal−digit character test

labs
: return the absolute value of a long integer

ldexp
: multiply floating−point number by integral power of 2

ldiv
: return quotient and remainder from division

localtime
: convert local date and time to ASCII

log
: calculate logarithm

log10

calculate logarithm base 10

longjmp
non−local jump

malloc
general memory allocation function

memchr
locate byte in byte string

memcmp
compare byte string

memcpy
copy byte string

memmove
copy overlapping byte string

memset
write a byte to byte string

mktime
convert broken down time to system time

modf
extract signed integral and fractional values from floating−point number

perror
write error messages to standard error

pow
calculate power

printf
formatted output conversion

putc
output a character to a stream

putchar
output a character to stdout

puts
output a line to stdout

qsort
sort function

raise
send a signal to the current process

rand
random number generator

realloc
reallocation of memory function

remove
remove directory entry

rewind
rewind stream

scanf
input format conversion

setbuf
stream buffering operations

setjmp
set up non−local jump

setvbuf
stream buffering operations

signal
: software signal facilities

sin
: sine function

sinh
: hyperbolic sine function

sprintf
: formatted output conversion

sqrt
: square root function

srand
: set random number seed

sscanf
: input format conversion

strcat
: concatenate strings

strchr
: locate character in string

strcmp
: compare strings

strcoll
: compare strings according to current collation

strcpy
: copy strings

strcspn
: span the complement of a string

strerror
: get error message string

strftime
: format date and time

strlen
: find length of string

strncat
: concatenate strings

strncmp
: compare strings

strncpy
: copy strings

strpbrk
: locate multiple characters in string

strrchr
: locate character in string

strspn
: span a string

strstr
: locate a substring in a string

strtod
: convert string to double

strtok
: string token operation

strtol

convert string value to a long integer

strtoul
convert a string to an unsigned long integer

strxfrm
transform a string under locale

system
pass a command to the shell

tan
tangent function

tanh
hyperbolic tangent function

time
get time of day

tmpfile
create a temporary file

tmpnam
create a temporary file name

tolower
upper case to lower case letter conversion

toupper
lower case to upper case letter conversion

ungetc
un−get character from input stream

va_arg
variable argument list

va_end
variable argument list

va_start
variable argument list

vfprintf
formatted output conversion

vprintf
formatted output conversion

vsprintf
formatted output conversion

assert − expression verification macro

SYNOPSIS

```
#include <assert.h>

assert(expression)
```

DESCRIPTION

The **assert**() macro tests the given *expression* and if it is false, the calling process is terminated. A diagnostic message is written to *stderr* and the abort function is called, effectively terminating the program.

If *expression* is true, the **assert**() macro does nothing.

The **assert**() macro may be removed at compile time by defining the preprocessor macro **NDEBUG**.

DIAGNOSTICS

The following diagnostic message is written to *stderr* if *expression* is false:

```
"Assert failed: %s, file %s, line %d\n", \
                    "expression", __FILE__, __LINE__);
```

SEE ALSO

abort

STANDARDS

The **assert**() macro conforms to ANSI X3.159−1989 (``ANSI C'').

HISTORY

A **assert** macro appeared in Version 6 AT&T UNIX.

ctype − character classification

SYNOPSIS

#include <***ctype.h***>
int ***isalnum****(int);*
Alphanumeric character test.
int ***isalpha****(int);*
Alphanetic character test.
int ***isascii****(int);*
Test for ASCII character.
int ***iscntrl****(int);*
Control character test.
int ***isdigit****(int);*
Decimal digit character test.
int ***isgraph****(int);*
Printing character test (excludes spaces).
int ***islower****(int);*
Lower case character test.
int ***isprint****(int);*
Printing character test (includes spaces).
int ***ispunct****(int);*
Punctuation character test.
int ***isspace****(int);*
Whitespace character test.
int ***isupper****(int);*
Upper case character test.
int ***isxdigit****(int);*
Hexadecimal digit character test.
int ***tolower****(int);*
Convert an upper case character to lower case.
int ***toupper****(int);*
Convert a lower case character to upper case.

DESCRIPTION

The above macros perform character tests and conversions on the integer argument.

See the specific manual pages for more information.

STANDARDS

These functions conform to ANSI X3.159−1989 (``ANSI C'').

isalnum − alphanumeric character test

SYNOPSIS

```
#include <ctype.h>

int
isalnum(int c)
```

DESCRIPTION

The **isalnum**() macro tests for any character for which isalpha or isdigit is true.

RETURN VALUES

The **isalnum**() macro returns zero if the character tests false and returns non−zero if the character tests true.

SEE ALSO

isalnum, isalpha, isascii, iscntrl, isdigit, isgraph, islower, isprint, ispunct, isspace, isupper, isxdigit, tolower, toupper, stdio

STANDARDS

The **isalnum**() function conforms to ANSI X3.159−1989 (``ANSI C'').

isalpha − alphabetic character test

SYNOPSIS

```
#include <ctype.h>

int
isalpha(int c)
```

DESCRIPTION

The **isalpha**() function tests for any character for which isupper or islower is true and for which none of iscntrl, isdigit, ispunct, or isspace is true. In the ``*C*'' locale, **isalpha**() returns true only for the characters for which isupper or islower is true.

RETURN VALUES

The **isalpha**() macro returns zero if the character tests false and returns non−zero if the character tests true.

SEE ALSO

isalnum, isascii, iscntrl, isdigit, isgraph, islower, isprint, ispunct, isspace, isupper, isxdigit, tolower, toupper, stdio

STANDARDS

The **isalpha**() function conforms to ANSI X3.159−1989 (``ANSI C'').

isascii − test for ASCII character

SYNOPSIS

```
#include <ctype.h>

int
isascii(int c)
```

DESCRIPTION

The **isascii**() function tests for an ASCII character, which is any character with a value less than than or equal to 0177.

SEE ALSO

isalnum, isalpha, iscntrl, isdigit, isgraph, islower, isprint, ispunct, isspace, isupper, isxdigit, tolower, toupper, stdio

STANDARDS

The **isascii**() function conforms to ANSI X3.159−1989 (``ANSI C'').

iscntrl − control character test

SYNOPSIS

```
#include <ctype.h>

int
iscntrl(int c)
```

DESCRIPTION

The **iscntrl**() function tests for any control character.

RETURN VALUES

The **iscntrl**() macro returns zero if the character tests false and returns non−zero if the character tests true.

SEE ALSO

isalnum, isalpha, isascii, isdigit, isgraph, islower, isprint, ispunct, isspace, isupper, isxdigit, tolower, toupper, stdio

STANDARDS

The **iscntrl**() function conforms to ANSI X3.159−1989 (``ANSI C'').

isdigit − decimal−digit character test

SYNOPSIS

```
#include <ctype.h>

int
isdigit(int c)
```

DESCRIPTION

The **isdigit**() function tests for any decimal−digit character.

RETURN VALUES

The **isdigit**() macro returns zero if the character tests false and returns non−zero if the character tests true.

SEE ALSO

isalnum, isalpha, isascii, iscntrl, isdigit, isgraph, islower, isprint, ispunct, isspace, isupper, isxdigit, tolower, toupper, stdio

STANDARDS

The **isdigit**() function conforms to ANSI X3.159−1989 (``ANSI C'').

isgraph − printing character test (space character exculsive)

SYNOPSIS

```
#include <ctype.h>

int
isgraph(int c)
```

DESCRIPTION

The **isgraph**() function tests for any printing character except space (' ').

RETURN VALUES

The **isgraph**() macro returns zero if the character tests false and returns non−zero if the character tests true.

SEE ALSO

isalnum, isalpha, isascii, iscntrl, isdigit, islower, isprint, ispunct, isspace, isupper, isxdigit, tolower, toupper, stdio

STANDARDS

The **isgraph**() function conforms to ANSI X3.159−1989 (``ANSI C'').

islower − lower−case character test

SYNOPSIS

```
#include <ctype.h>

int
islower(int c)
```

DESCRIPTION

The **islower**() function tests for any lower−case letter for which none of iscntrl, isdigit, ispunct, or isspace is true. In the ``*C*'' locale, **islower**() returns true only for the characters defined as lower−case letters.

RETURN VALUES

The **islower**() macro returns zero if the character tests false and returns non−zero if the character tests true.

SEE ALSO

isalnum, isalpha, isascii, iscntrl, isdigit, isgraph, isprint, ispunct, isspace, isupper, isxdigit, tolower, toupper, stdio

STANDARDS

The **islower**() function conforms to ANSI X3.159−1989 (``ANSI C'').

isprint − printing character test (space character inclusive)

SYNOPSIS

```
#include <ctype.h>

int
isprint(int c)
```

DESCRIPTION

The **isprint**() function tests for any printing character including space (' ').

RETURN VALUES

The **isprint**() macro returns zero if the character tests false and returns non−zero if the character tests true.

SEE ALSO

isalnum, isalpha, isascii, iscntrl, isdigit, isgraph, islower, ispunct, isspace, isupper, isxdigit, tolower, toupper, stdio

STANDARDS

The **isprint**() function conforms to ANSI X3.159−1989 (``ANSI C'').

ispunct − punctuation character test

SYNOPSIS

```
#include <ctype.h>

int
ispunct(int c)
```

DESCRIPTION

The **ispunct**() function tests for any printing character except space (' ') or a character for which isalnum is true.

RETURN VALUES

The **ispunct**() macro returns zero if the character tests false and returns non−zero if the character tests true.

SEE ALSO

isalnum, isalpha, isascii, iscntrl, isdigit, isgraph, islower, isprint, isspace, isupper, isxdigit, tolower, toupper, stdio

STANDARDS

The **ispunct**() function conforms to ANSI X3.159−1989 (``ANSI C'').

isspace − white−space character test

SYNOPSIS

```
#include <ctype.h>

int
isspace(int c)
```

DESCRIPTION

The **isspace**() function tests for the standard white−space characters for which isalnum is false. The standard white−space characters are the following:

```
` '    Space character.
\f     Form feed.
\n     New−line.
\r     Carriage return.
\t     Horizontal tab.
\v     And vertical tab.
```

In the ``*C*'' locale, **isspace**() returns true only for the standard whitespace characters.

RETURN VALUES

The **isspace**() macro returns zero if the character tests false and returns non−zero if the character tests true.

SEE ALSO

isalnum, isalpha, isascii, iscntrl, isdigit, isgraph, islower, isprint, ispunct, isupper, isxdigit, tolower, toupper, stdio

STANDARDS

The **isspace**() function conforms to ANSI X3.159−1989 (``ANSI C'').

isupper − upper−case character test

SYNOPSIS

```
#include <ctype.h>

int
isupper(int c)
```

DESCRIPTION

The **isupper**() function tests for any upper−case letter or any of an implementation−defined set of characters for which none of iscntrl, isdigit, ispunct, or isspace is true. In the ``*C*'' locale, **isupper**() returns true only for the characters defined as upper−case letters.

RETURN VALUES

The **isupper**() macro returns zero if the character tests false and returns non−zero if the character tests true.

SEE ALSO

isalnum, isalpha, isascii, iscntrl, isdigit, isgraph, islower, isprint, ispunct, isspace, isxdigit, tolower, toupper, stdio

STANDARDS

The isupper function conforms to ANSI X3.159−1989 (``ANSI C'').

isxdigit − hexadecimal−digit character test

SYNOPSIS

```
#include <ctype.h>

int
isxdigit(int c)
```

DESCRIPTION

The **isxdigit**() function tests for any hexadecimal−digit character.

RETURN VALUES

The **isxdigit**() macro returns zero if the character tests false and returns non−zero if the character tests true.

SEE ALSO

isalnum, isalpha, isascii, iscntrl, isdigit, isgraph, islower, isprint, ispunct, isspace, isupper, tolower, toupper, stdio

STANDARDS

The **isxdigit**() function conforms to ANSI X3.159−1989 (``ANSI C'').

tolower − upper case to lower case letter conversion

SYNOPSIS

```
#include <ctype.h>

int
tolower(int c)
```

DESCRIPTION

The **tolower**() function converts an upper−case letter to the corresponding lower−case letter.

RETURN VALUES

If the argument is an upper−case letter, the **tolower**() function returns the corresponding lower−case letter if there is one; otherwise the argument is returned unchanged.

SEE ALSO

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

STANDARDS

The **tolower**() function conforms to ANSI X3.159−1989 (``ANSI C'').

toupper − lower case to upper case letter conversion

SYNOPSIS

```
#include <ctype.h>

int
toupper(int c)
```

DESCRIPTION

The **toupper**() function converts a lower−case letter to the corresponding upper−case letter. If the argument is a lower−case letter, the **toupper**() function returns the corresponding upper−case letter if there is one; otherwise the argument is returned unchanged.

SEE ALSO

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

STANDARDS

The **toupper**() function conforms to ANSI X3.159−1989 (``ANSI C'').

errno − error numbers

SYNOPSIS

```
#include <errno.h>
```

DESCRIPTION

This section describes error numbers.

DIAGNOSTICS

Some library functions provide an error number in the external variable *errno*, which is defined as:

```
extern int errno
```

When a library function detects an error, it returns an integer value indicating failure (usually −1) and sets the variable *errno* accordingly. (This allows interpretation of the failure on receiving a −1 and to take action accordingly.) Successful calls never set *errno*; once set, it remains until another error occurs. It should only be examined after an error.

The following is a complete list of the errors and their names as given in <*sys/errno.h*>.

−100 EDOM
: *Numerical argument out of domain*. A numerical input argument was outside the defined domain of the mathematical function.

−101 ERANGE
: *Numerical result out of range*. A numerical result of the function was too large to fit in the available space (perhaps exceeded precision).

SEE ALSO

perror

locale− localization

SYNOPSIS

#include <locale.h>

math − mathematics

SYNOPSIS

#include <***math.h***>

double **acos**(*double*)**;**
Arc cosine function.
double ***asin****(double);*
Arc sine function.
double ***atan****(double);*
Arc tangent function.
double ***atan2****(double, double);*
double ***ceil****(double);*
Round to smallest integral value greater than or equal to.
double ***cos****(double);*
Cosine function.
double ***cosh****(double);*
Hyperbolc cosine function.
double ***exp****(double);*
Exponentiation function.
double ***fabs****(double);*
Floating point absolute value function.
double ***floor****(double);*
Round to largets integral value not greater than.
double ***fmod****(double, double);*
Floating point remainder function.
double ***frexp****(double, int *);*
Convert floating point number to fractional and integral components.
double ***ldexp****(double, int);*
Multiply a floating point number by an integral power of two.
double ***log****(double);*
Natural logarithm function.
double ***log10****(double);*
Base 10 logarithm function.
double ***modf****(double, double *);*
Extract signed integral and fractional parts from a floating point number.
double ***pow****(double, double);*
Compute the first argument to the power of the second.
double ***sin****(double);*
Sine function.
double ***sinh****(double);*
Hyperbolic sine function.
double ***sqrt****(double);*
Square root function.
double ***tan****(double);*
Tangent function.
double ***tanh****(double);*
Hyperbolic tangent function.

acos − arc cosine function

SYNOPSIS

```
#include <math.h>

double
acos(double x)
```

DESCRIPTION

The **acos**() function computes the principal value of the arc cosine of *x* in the range [0, pi].

RETURN VALUES

Returns the arc cosine of *x*.

SEE ALSO

sin, cos, tan, asin, atan, atan2, sinh, cosh, tanh

asin − arc sine function

SYNOPSIS

```
#include <math.h>

double
asin(double x)
```

DESCRIPTION

The **asin**() function computes the principal value of the arc sine of *x* in the range [−pi/2, +pi/2].

RETURN VALUES

Returns the arc sine of *x*.

SEE ALSO

acos, atan, atan2, cos, cosh, sin, sinh, tan, tanh

atan − arc tangent function of one variable

SYNOPSIS

```
#include <math.h>

double
atan(double x)
```

DESCRIPTION

The **atan**() function computes the principal value of the arc tangent of *x* in the range [−pi/2, +pi/2].

RETURN VALUES

Returns the arc tabgent of *x*.

SEE ALSO

acos, asin, atan2, cos, cosh, sin, sinh, tan, tanh

atan2 − arc tangent function of two variables

SYNOPSIS

```
#include <math.h>

double
atan2(double y, double x)
```

DESCRIPTION

The **atan2**() function computes the principal value of the arc tangent of *y/x*, using the signs of both arguments to determine the quadrant of the return value.

RETURN VALUES

The atan2 function, if successful, returns the arc tangent of *y/x* in the range [−pi, +pi] radians. If both *x* and *y* are zero, the global variable *errno* is set to **EDOM**.

```
atan2(y, x):= atan(y/x)                          if x > 0,
              sign(y)*(pi − atan(|y/x|))         if x < 0,
              0                                  if x = y = 0, or
              sign(y)*pi/2                       if x = 0 y.
```

NOTES

The function **atan2**() defines "if x > 0," **atan2**(*0*, *0*) = 0 despite that previously **atan2**(*0*, *0*) may have generated an error message. The reasons for assigning a value to **atan2**(*0*, *0*) are these:

1. Programs that test arguments to avoid computing **atan2**(*0*, *0*) must be indifferent to its value. Programs that require it to be invalid are vulnerable to diverse reactions to that inva− lidity on diverse computer systems.
2. The **atan2**() function is used mostly to convert from rectangular (x,y) to polar (r,theta) coordinates that must satisfy x = r*cos theta and y = r*sin theta. These equations are satisfied when (x=0,y=0) is mapped to (r=0,theta=0). In general, conversions to polar coordinates should be computed thus:

   ```
   r      := hypot(x,y);  ... := sqrt(x*x+y*y)
   theta  := atan2(y,x).
   ```

3. The foregoing formulas need not be altered to cope in a reasonable way with signed zeros and infinities on a machine that conforms to IEEE 754; **atan2**() provides for such a machine are designed to handle all cases. That is why **atan2**(+−*0*, −*0*) = +−pi for instance. In general the formulas above are equivalent to these:

   ```
   r := sqrt(x*x+y*y); if r = 0 then x := copysign(1,x);
   ```

SEE ALSO

acos, asin, atan, cos, cosh, sin, sinh, tan, tanh

STANDARDS

The **atan2**() function conforms to ANSI X3.159−1989 (``ANSI C").

ceil − round to smallest integral value greater than or equal

SYNOPSIS

```
#include <math.h>

double
ceil(double x)
```

DESCRIPTION

The **ceil**() function returns the smallest integral value greater than or equal to *x*.

SEE ALSO

abs, fabs, floor

STANDARDS

The **ceil**() function conforms to ANSI X3.159−1989 (``ANSI C'').

cos − cosine function

SYNOPSIS

```
#include <math.h>

double
cos(double x)
```

DESCRIPTION

The **cos**() function computes the cosine of *x* (measured in radians). A large magnitude argument may yield a result with little or no significance.

RETURN VALUES

The **cos**() function returns the cosine value.

SEE ALSO

sin, tan, asin, acos, atan, atan2, sinh, cosh, tanh

STANDARDS

The **cos**() function conforms to ANSI X3.159−1989 (``ANSI C'').

cosh − hyperbolic cosine function

SYNOPSIS

```
#include <math.h>

double
cosh(double x)
```

DESCRIPTION

The **cosh**() function computes the hyperbolic cosine of *x*.

RETURN VALUES

Returns the hyperbolic cosine of *x*.

SEE ALSO

acos, asin, atan, atan2, cos, sin, sinh, tan, tanh

exp, log, log10, pow − exponential, logarithm, power functions

SYNOPSIS

```
#include <math.h>

double
exp(double x)
double
log(double x)
double
log10(double x)
double
pow(double x, double y)
```

DESCRIPTION

The **exp**() function computes the exponential value of the given argument *x*.

The **log**() function computes the value of the natural logarithm of argument *x*.

The **log10**() function computes the value of the logarithm of argument *x* to base 10.

The **pow**() computes the value of *x* to the exponent *y*.

ERROR (due to Roundoff etc.)

exp(x) and log(x) are accurate to within an *ulp*, and log10(x) to within about 2 *ulps*; an *ulp* is one *Unit* in the *Last Place*. The error in **pow**(*x*, *y*) is below about 2 *ulps* when its magnitude is moderate, but increases as **pow**(*x*, *y*) approaches the over/underflow thresholds until almost as many bits could be lost as are occupied by the floating−point format's exponent field; that is 11 bits for IEEE 754 Double. No such drastic loss has been exposed by testing; the worst errors observed have been below 300 *ulps* for IEEE 754 Double. Moderate values of **pow**() are accurate enough that **pow**(*integer*, *integer*) is exact until it is bigger than 2**53 for IEEE 754.

RETURN VALUES

These functions will return the appropriate computation unless an error occurs or an argument is out of range. The functions **exp**() and **pow**() detect if the computed value will overflow and set the global variable *errno to* ERANGE. The function **pow**(*x*, *y*) checks to see if *x* < 0 and *y* is not an integer, in the event this is true, the global variable *errno* is set to EDOM.

NOTES

The function **pow**(*x*, *0*) returns x**0 = 1 for all x including x = 0, Infinity, and *NaN*. Previous implementations of pow may have defined x**0 to be undefined in some or all of these cases. Here are reasons for returning x**0 = 1 always:

1. Any program that already tests whether x is zero (or infinite or *NaN*) before computing x**0 cannot care whether 0**0 = 1 or not. Any program that depends upon 0**0 to be invalid is dubious anyway since that expression's meaning and, if invalid, its consequences vary from one computer system to another.
2. Some Algebra texts (e.g. Sigler's) define x**0 = 1 for all x, including x = 0. This is compatible with the convention that accepts a[0] as the value of polynomial

```
p(x) = a[0]*x**0 + a[1]*x**1 + a[2]*x**2 +...+ a[n]*x**n
```

at x = 0 rather than reject a[0]*0**0 as invalid.

3. Analysts will accept 0**0 = 1 despite that x**y can approach anything or nothing as x and y approach 0 independently. The reason for setting 0**0 = 1 anyway is this:
   If x(z) and y(z) are *any* functions analytic (expandable in power series) in z around z = 0, and if there x = y = 0, then x(z)**y(z) −> 1 as z −> 0.
4. If 0**0 = 1, then infinity**0 = 1/0**0 = 1 too; and then *NaN***0 = 1 too because x**0 = 1 for all finite and infinite x, i.e., independently of x.

HISTORY

A **exp**(), **log**() and **pow**() functions appeared in Version 6 AT&T UNIX. A **log10**() function appeared in Version 7 AT&T UNIX.

fabs − floating−point absolute value function

SYNOPSIS

```
#include <math.h>

double
fabs(double x)
```

DESCRIPTION

The **fabs**() function computes the absolute value of a floating−point number *x*.

RETURN VALUES

The **fabs**() function returns the absolute value of *x*.

SEE ALSO

abs, ceil, floor

STANDARDS

The **fabs**() function conforms to ANSI X3.159−1989 (``ANSI C'').

floor − round to largest integral value not greater than x

SYNOPSIS

```
#include <math.h>

double
floor(double x)
```

DESCRIPTION

The **floor**() function returns the largest integral value less than or equal to *x*.

SEE ALSO

abs, ceil, fabs

STANDARDS

The **floor**() function conforms to ANSI X3.159−1989 (``ANSI C'').

fmod − floating−point remainder function

SYNOPSIS

```
#include <math.h>

double
fmod(double x, double y)
```

DESCRIPTION

The **fmod**() function computes the floating−point remainder of *x*/ *y*.

RETURN VALUES

The **fmod**() function returns the value *x*−*i***y*, for some integer *i* such that, if *y* is non−zero, the result has the same sign as *x* and magnitude less than the magnitude of *y*. If *y* is zero, whether a domain error occurs or the **fmod**() function returns zero is implementation−defined.

SEE ALSO

STANDARDS

The **fmod**() function conforms to ANSI X3.159−1989 (``ANSI C'').

frexp − convert floating−point number to fractional and integral components

SYNOPSIS

```
#include <math.h>

double
frexp(double value, int *exp)
```

DESCRIPTION

The **frexp**() function breaks a floating−point number into a normalized fraction and an integral power of 2. It stores the integer in the *int* object pointed to by *exp*.

RETURN VALUES

The **frexp**() function returns the value *x*, such that *x* is a *double* with magnitude in the interval [1/2, 1] or zero, and *value* equals *x* times 2 raised to the power **exp*. If *value* is zero, both parts of the result are zero.

SEE ALSO

ldexp, modf, math

STANDARDS

The **frexp**() function conforms to ANSI X3.159−1989 (``ANSI C'').

ldexp − multiply floating−point number by integral power of 2

SYNOPSIS

```
#include <math.h>

double
ldexp(double x, int exp)
```

DESCRIPTION

The **ldexp**() function multiplies a floating−point number by an integral power of 2.

RETURN VALUES

The **ldexp**() function returns the value of *x* times 2 raised to the power *exp*.

If the resultant value would cause an overflow, the global variable *errno* is set to ERANGE and the value HUGE is returned.

SEE ALSO

frexp, modf, math

STANDARDS

The **ldexp**() function conforms to ANSI X3.159−1989 (``ANSI C'').

modf − extract signed integral and fractional values from floating−point number

SYNOPSIS

```
#include <math.h>

double
modf(double value, double *iptr)
```

DESCRIPTION

The **modf**() function breaks the argument *value* into integral and fractional parts, each of which has the same sign as the argument. It stores the integral part as a *double* in the object pointed to by *iptr*.

RETURN VALUES

The **modf**() function returns the signed fractional part of *value*.

SEE ALSO

frexp, ldexp, math

STANDARDS

The **modf**() function conforms to ANSI X3.159−1989 (``ANSI C'').

sin − sine function

SYNOPSIS

```
#include <math.h>

double
sin(double x)
```

DESCRIPTION

The **sin**() function computes the sine of *x* (measured in radians). A large magnitude argument may yield a result with little or no significance.

RETURN VALUES

The **sin**() function returns the sine value.

SEE ALSO

acos, asin, atan, atan2, cos, cosh, sinh, tan, tanh, math

STANDARDS

The **sin**() function conforms to ANSI X3.159−1989 (``ANSI C'').

sinh − hyperbolic sine function

SYNOPSIS

```
#include <math.h>

double
sinh(double x)
```

DESCRIPTION

The **sinh**() function computes the hyperbolic sine of *x*.

RETURN VALUES

The **sinh**() function returns the hyperbolic sine value unless the magnitude of *x* is too large; in this event, the global variable *errno* is set to ERANGE.

SEE ALSO

acos, asin, atan, atan2, cos, cosh, sin, tan, tanh, math

STANDARDS

The **sinh**() function conforms to ANSI X3.159−1989 (``ANSI C'').

sqrt − square root function

SYNOPSIS

```
#include <math.h>

double
sqrt(double x)
```

DESCRIPTION

The **sqrt**() function computes the non−negative square root of *x*.

RETURN VALUES

Returns the square root of *x*.

SEE ALSO

math

tan − tangent function

SYNOPSIS

```
#include <math.h>

double
tan(double x)
```

DESCRIPTION

The **tan**() function computes the tangent of *x* (measured in radians). A large magnitude argument may yield a result with little or no significance. For a discussion of error due to roundoff, see math.

RETURN VALUES

The **tan**() function returns the tangent value.

SEE ALSO

acos, asin, atan, atan2, cos, cosh, sin, sinh, tanh, math

STANDARDS

The **tan**() function conforms to ANSI X3.159−1989 (``ANSI C'').

tanh − hyperbolic tangent function

SYNOPSIS

```
#include <math.h>

double
tanh(double x)
```

DESCRIPTION

The **tanh**() function compute the hyperbolic tangent of *x*. For a discussion of error due to roundoff, see math.

RETURN VALUES

The **tanh**() function returns the hyperbolic tangent value.

SEE ALSO

acos, asin, atan, atan2, cos, cosh, sin, sinh, tan, math

STANDARDS

The **tanh**() function conforms to ANSI X3.159−1989 (``ANSI C'').

setjmp, longjmp − non−local jumps

SYNOPSIS

```
#include <setjmp.h>

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

DESCRIPTION

The **setjmp**() function saves its calling environment in *env*. This function returns 0.

The **longjmp**() function restores the environment saved by its most recent invocation of the **setjmp**() function. It then returns so that program execution continues as if the corresponding invocation of the **setjmp**() call had just returned the value specified by *val*, instead of 0.

The **longjmp**() routines may not be called after the routine which called the **setjmp**() routines returns.

All accessible objects have values as of the time **longjmp**() routine was called, except that the values of objects of automatic storage invocation duration and have been changed between the **setjmp**() invocation and **longjmp**() call are indeterminate.

STANDARDS

The **setjmp**() and **longjmp**() functions conform to ANSI X3.159−1989 (``ANSI C'').

signal − software signal facilities

SYNOPSIS

```
#include <signal.h>

int
raise (int sig)
void
(*signal(int sig, void (*func)(int)))(int)
```

DESCRIPTION

Signal is not yet implemented.

RETURN VALUES

The previous action is returned on a successful call. Otherwise, −1 is returned and the global variable *errno* is set to indicate the error.

SEE ALSO

setjmp

stdarg − variable argument lists

SYNOPSIS

```
#include <stdarg.h>

void
va_start(va_list ap, last)
type
va_arg(va_list ap, type)
void
va_end(va_list ap)
```

DESCRIPTION

A function may be called with a varying number of arguments of varying types. The include file <*stdarg.h*> declares a type (*va_list*) and defines three macros for stepping through a list of arguments whose number and types are not known to the called function.

The called function must declare an object of type *va_list* which is used by the macros **va_start**(), **va_arg**(), and **va_end**().

The **va_start**() macro initializes *ap* for subsequent use by **va_arg**() and **va_end**(), and must be called first.

The parameter *last* is the name of the last parameter before the variable argument list, i.e. the last parameter of which the calling function knows the type.

Because the address of this parameter is used in the **va_start**() macro, it should not be declared as a register variable, or as a function or an array type.

The **va_start**() macro returns no value.

The **va_arg**() macro expands to an expression that has the type and value of the next argument in the call. The parameter *ap* is the *va_list ap* initialized by **va_start**(). Each call to **va_arg**() modifies *ap* so that the next call returns the next argument. The parameter *type* is a type name specified so that the type of a pointer to an object that has the specified type can be obtained simply by adding a * to *type*.

If there is no next argument, or if *type* is not compatible with the type of the actual next argument (as promoted according to the default argument promotions), random errors will occur.

The first use of the **va_arg**() macro after that of the **va_start**() macro returns the argument after *last*. Successive invocations return the values of the remaining arguments.

The **va_end**() macro handles a normal return from the function whose variable argument list was initialized by **va_start**().

The **va_end**() macro returns no value.

EXAMPLES

The function *foo* takes a string of format characters and prints out the argument associated with each format character based on the type.

```
void foo(char *fmt, ...)
{
        va_list ap;
        int d;
        char c, *p, *s;

        va_start(ap, fmt);
        while (*fmt)
                switch(*fmt++) {
                case 's':                       /* string */
                        s = va_arg(ap, char *);
                        printf("string %s\n", s);
                        break;
                case 'd':                       /* int */
                        d = va_arg(ap, int);
                        printf("int %d\n", d);
                        break;
                case 'c':                       /* char */
                        c = va_arg(ap, char);
                        printf("char %c\n", c);
                        break;
                }
        va_end(ap);
}
```

STANDARDS

The **va_start**(), **va_arg**(), and **va_end**() macros conform to ANSI X3.159−1989 (``ANSI C").

stddef − standard definitions

SYNOPSIS

#include <stddef.h>

stdio − standard input/output library functions

SYNOPSIS

#include <***stdio.h***>

FILE *stdin;
FILE *stdout;
FILE *stderr;

void ***clearerr****(FILE *);*
Clear the end−of−file and error indicators on a stream.

int ***fclose****(FILE *);*
Close a stream.

int ***feof****(FILE *);*
Test for end−of−file on a stream.

int ***ferror****(FILE *);*
Test for error on a stream.

int ***fflush****(FILE *);*
Flush a stream.

int ***fgetc****(FILE *);*
Get the next character from a stream.

int ***fgetpos****(FILE *, fpos_t *);*
Get the current stream position.

*char **fgets(char *, size_t, FILE *);*
Get a line from a stream.

*FILE **fopen(char *, char *);*
Open a stream.

int ***fprintf****(FILE *, const char *, ...);*
Formatted output to a stream.

int ***fputc****(int, FILE *);*
Output a character to a stream.

int ***fputs****(const char *, FILE *);*
Output a line to a stream.

int ***fread****(void *, size_t, size_t, FILE *);*
Binary stream input.

*FILE **freopen(char *, char *, FILE *);*
Reopen a stream.

int ***fscanf****(FILE *, const char *, ...);*
Formatted input from a stream.

int ***fseek****(FILE *, long, int);*
Get the current stream position.

int ***fsetpos****(FILE *, fpos_t);*
Set a position in a stream.

long ***ftell****(FILE *);*
Get the current stream position.

size_t ***fwrite****(const void *, size_t, size_t, FILE *);*
Binary stream output.

int ***getc****(FILE *);*
Get the next character from a stream.

int ***getchar****();*
Get the next character from *stdin*.

*char ***gets**(char *);*
Get a line from *stdin*.
*void **perror**(const char *);*
Write error messages to stderr.
*int **printf**(const char *, ...);*
Formatted output to *stdout*.
*int **putc**(int, FILE *);*
Output a character to a stream.
*int **putchar**(int);*
Output a character to *stdout*.
*int **puts**(const char *);*
Output a line to *stdout*.
*int **remove**(const char *);*
Remove a directory entry.
*void **rewind**(FILE *);*
Set the position in a stream to its beginning.
*int **scanf**(const char *, ...);*
Formatted input from *stdin*.
*void **setbuf**(FILE *, char *);*
Set a buffer for a stream.
*int **setvbuf**(FILE *, char *, int, size_t);*
Control buffering on a stream.
*int **sprintf**(char *, const char *, ...);*
Formatted output to a string.
*int **sscanf**(const char *, const char *, ...);*
Formatted input from a string.
*FILE ***tmpfile**(void);*
Create a temporary file.
*char ***tmpname**(char *);*
Create a temporary file name.
*int **ungetc**(int, FILE *);*
Push a character back to the input stream.
*int **vfprintf**(FILE *, const char *, va_list);*
Variable argument formatted output to a stream.
*int **vprintf**(const char *, va_list);*
Variable argument formatted output to *stdout*.
*int **vsprintf**(char *, const char *, va_list);*
Variable argument formatted output to a string.

DESCRIPTION

The standard I/O library provides a simple and efficient buffered stream I/O interface. Input and output is mapped into logical data streams and the physical I/O characteristics are concealed. The functions and macros are listed below; more information is available from the individual man pages.

A stream is associated with an external file (which may be a physical device) by *opening* a file, which may involve creating a new file. Creating an existing file causes its former contents to be discarded. If a file can support positioning requests (such as a disk file, as opposed to a terminal) then a *file position indicator* associated with the stream is positioned at the start of the file (byte zero), unless the file is opened with append mode. If append mode is used, the position indicator will be placed the end−of−file. The position indicator is maintained by subsequent reads, writes and

positioning requests. All input occurs as if the characters were read by successive calls to the fgetc function; all output takes place as if all characters were read by successive calls to the fputc function.

A file is disassociated from a stream by *closing* the file. Output streams are flushed (any unwritten buffer contents are transferred to the host environment) before the stream is disassociated from the file. The value of a pointer to a FILE object is indeterminate after a file is closed (garbage).

A file may be subsequently reopened, by the same or another program execution, and its contents reclaimed or modified (if it can be repositioned at the start). If the main function returns to its original caller, or the exit function is called, all open files are closed (hence all output streams are flushed) before program termination. Other methods of program termination, such as abort do not bother about closing files properly.

This implementation needs and makes no distinction between ``text'' and ``binary'' streams. In effect, all streams are binary. No translation is performed and no extra padding appears on any stream.

At program startup, three streams are predefined and need not be opened explicitly:

o *standard input* (for reading conventional input),
o *standard output* (for writing conventional output), and
o *standard error* (for writing diagnostic output). These streams are abbreviated *stdin*, *stdout* and *stderr*. Initially, the standard error stream is unbuffered; the standard input and output streams are fully buffered if and only if the streams do not refer to an interactive or ``terminal'' device. In these cases, or when a large amount of computation is done after printing part of a line on an output terminal, it is necessary to fflush the standard output before going off and computing so that the output will appear. Alternatively, these defaults may be modified via the setvbuf function. **STANDARDS**

The **stdio** library conforms to ANSI X3.159−1989 (``ANSI C'').

clearerr, feof, ferror − check and reset stream status

SYNOPSIS

```
#include <stdio.h>

void
clearerr(FILE *stream)
int
feof(FILE *stream)
int
ferror(FILE *stream)
```

DESCRIPTION

The function **clearerr**() clears the end−of−file and error indicators for the stream pointed to by *stream*.

The function **feof**() tests the end−of−file indicator for the stream pointed to by *stream*, returning non−zero if it is set. The end−of−file indicator can only be cleared by the function **clearerr**().

The function **ferror**() tests the error indicator for the stream pointed to by *stream*, returning non−zero if it is set. The error indicator can only be reset by the **clearerr**() function.

ERRORS

These functions should not fail and do not set the external variable *errno*.

SEE ALSO

stdio

STANDARDS

The functions **clearerr**(), **feof**(), and **ferror**() conform to ANSI X3.159−1989 (``ANSI C'').

fclose − close a stream

SYNOPSIS

```
#include <stdio.h>

int
fclose(FILE *stream)
```

DESCRIPTION

The **fclose**() function dissociates the named *stream* from its underlying file or set of functions. If the stream was being used for output, any buffered data is written first, using fflush.

RETURN VALUES

Upon successful completion 0 is returned. Otherwise, EOF is returned and the global variable *errno* is set to indicate the error. In either case no further access to the stream is possible.

ERRORS

[EBADF]
The argument *stream* is not an open stream.

The **fclose**() function may also fail and set *errno* for any of the errors specified for fflush.

SEE ALSO

fflush, fopen, setbuf

STANDARDS

The **fclose**() function conforms to ANSI X3.159−1989 (``ANSI C'').

fflush − flush a stream

SYNOPSIS

```
#include <stdio.h>

int
fflush(FILE *stream)
```

DESCRIPTION

The function **fflush**() forces a write of all buffered data for the given output or update *stream* via the stream's underlying write function. The open status of the stream is unaffected.

If the *stream* argument is NULL, **fflush**() flushes *all* open output streams.

RETURN VALUES

Upon successful completion 0 is returned. Otherwise, EOF is returned and the global variable *errno* is set to indicate the error.

ERRORS

[EBADF]
Stream is not an open stream, or not a stream open for writing.

SEE ALSO

fopen, fclose, setbuf

STANDARDS

The **fflush**() function conforms to ANSI X3.159−1989 (``ANSI C'').

fgetc, getc, getchar − get next character from input stream

SYNOPSIS

```
#include <stdio.h>

int
fgetc(FILE *stream)
int
getc(FILE *stream)
int
getchar()
```

DESCRIPTION

The **fgetc**() function obtains the next input character (if present) from the stream pointed at by *stream*, or the next character pushed back on the stream via ungetc.

The **getc**() function acts essentially identically to **fgetc**(), but is a macro that expands in−line.

The **getchar**() function is equivalent to: getc with the argument stdin.

RETURN VALUES

If successful, these routines return the next requested object from the *stream*. If the stream is at end−of−file or a read error occurs, the routines return EOF. The routines feof and ferror must be used to distinguish between end−of−file and error. If an error occurs, the global variable *errno* is set to indicate the error. The end−of−file condition is remembered, even on a terminal, and all subsequent attempts to read will return EOF until the condition is cleared with clearerr.

SEE ALSO

ferror, fread, fopen, putc, ungetc

STANDARDS

The **fgetc**(), **getc**() and **getchar**() functions conform to ANSI X3.159−1989 (``ANSI C'').

fgetpos, fseek, fsetpos, ftell, rewind − reposition a stream

SYNOPSIS

```
#include <stdio.h>

int
fseek(FILE *stream, long offset, int whence)
long
ftell(FILE *stream)
void
rewind(FILE *stream)
int
fgetpos(FILE *stream, fpos_t *pos)
int
fsetpos(FILE *stream, fpos_t *pos)
```

DESCRIPTION

The **fseek**() function sets the file position indicator for the stream pointed to by *stream*. The new position, measured in bytes, is obtained by adding *offset* bytes to the position specified by *whence*. If *whence* is set to SEEK_SET, SEEK_CUR, or SEEK_END, the offset is relative to the start of the file, the current position indicator, or end−of−file, respectively. A successful call to the **fseek**() function clears the end−of−file indicator for the stream and undoes any effects of the ungetc function on the same stream.

The **ftell**() function obtains the current value of the file position indicator for the stream pointed to by *stream*.

The **rewind**() function sets the file position indicator for the stream pointed to by *stream* to the beginning of the file. It is equivalent to:

```
(void)fseek(stream, 0L, SEEK_SET)
```

except that the error indicator for the stream is also cleared (see clearerr).

The **fgetpos**() and **fsetpos**() functions are alternate interfaces equivalent to **ftell**() and **fseek**() (with whence set to SEEK_SET), setting and storing the current value of the file offset into or from the object referenced by *pos*.

RETURN VALUES

The **rewind**() function returns no value. Upon successful completion, **fgetpos**(), **fseek**(), **fsetpos**() return 0, and **ftell**() returns the current offset. Otherwise, **fseek**() returns −1 and the others return a nonzero value and the global variable *errno* is set to indicate the error.

ERRORS

[EBADF]
The *stream* specified is not a seekable stream.

[EINVAL]
The *whence* argument to **fseek**() was not SEEK_SET, SEEK_END, or SEEK_CUR.

The function **fgetpos**(), **fseek**(), **fsetpos**(), and **ftell**() may also fail and set *errno* for any of the errors

specified for the routines fflush, and malloc.

SEE ALSO

STANDARDS

The **fgetpos**(), **fsetpos**(), **fseek**(), **ftell**(), and **rewind**() functions conform to ANSI X3.159−1989 (``ANSI C'').

fgets, gets − get a line from a stream

SYNOPSIS

```
#include <stdio.h>

char *
fgets(char *str, size_t size, FILE *stream)
char *
gets(char *str)
```

DESCRIPTION

The **fgets**() function reads at most one less than the number of characters specified by size from the given *stream* and stores them in the string *str*. Reading stops when a newline character is found, at end−of−file or error. The newline, if any, is retained. In any case a `\0' character is appended to end the string.

The **gets**() function is equivalent to **fgets**() with an infinite size and a *stream* of *stdin*, except that the newline character (if any) is not stored in the string. It is the caller's responsibility to ensure that the input line, if any, is sufficiently short to fit in the string.

RETURN VALUES

Upon successful completion, **fgets**() and **gets**() return a pointer to the string. If end−of−file or an error occurs before any characters are read, they return NULL. The **fgets**() and functions **gets**() do not distinguish between end−of−file and error, and callers must use feof and ferror to determine which occurred.

ERRORS

[EBADF]
The given *stream* is not a readable stream.

The function **fgets**() may also fail and set *errno* for any of the errors specified for the routines fflush or malloc.

The function **gets**() may also fail and set *errno* for any of the errors specified for the routine getchar.

SEE ALSO

feof, ferror

STANDARDS

The functions **fgets**() and **gets**() conform to ANSI X3.159−1989 (``ANSI C'').

BUGS

Since it is usually impossible to ensure that the next input line is less than some arbitrary length, and because overflowing the input buffer is almost invariably a security violation, programs should *NEVER* use **gets**(). The **gets**() function exists purely to conform to ANSI X3.159−1989 (``ANSI C'').

fopen, freopen − stream open functions

SYNOPSIS

```
#include <stdio.h>

FILE *
fopen(char *path, char *mode)
FILE *
freopen(char *path, char *mode, FILE *stream)
```

DESCRIPTION

The **fopen**() function opens the file whose name is the string pointed to by *path* and associates a stream with it.

The argument *mode* points to a string beginning with one of the following sequences (Additional characters may follow these sequences.):

r
: Open text file for reading. The stream is positioned at the beginning of the file.

r+
: Open for reading and writing. The stream is positioned at the beginning of the file.

w
: Truncate file to zero length or create text file for writing. The stream is positioned at the beginning of the file. It ``w+" Open for reading and writing. The file is created if it does not exist, otherwise it is truncated. The stream is positioned at the beginning of the file.

a
: Open for writing. The file is created if it does not exist. The stream is positioned at the end of the file.

a+
: Open for reading and writing. The file is created if it does not exist. The stream is positioned at the end of the file.

The *mode* string can also include the letter ``b" either as a third character or as a character between the characters in any of the two−character strings described above. This is strictly for compatibility with ANSI X3.159−1989 (``ANSI C") and has no effect; the ``b" is ignored.

The **freopen**() function opens the file whose name is the string pointed to by *path* and associates the stream pointed to by *stream* with it. The original stream (if it exists) is closed. The *mode* argument is used just as in the fopen function. The primary use of the **freopen**() function is to change the file associated with a standard text stream (*stderr*, *stdin*, or *stdout*).

RETURN VALUES

Upon successful completion **fopen**() and **freopen**() return a FILE pointer. Otherwise, NULL is returned and the global variable *errno* is set to indicate the error.

ERRORS

[EINVAL]
: The *mode* provided to **fopen**() or **freopen**() was invalid.

The **fopen**() and **freopen**() functions may also fail and set *errno* for any of the errors specified for the routine malloc.

The **freopen**() function may also fail and set *errno* for any of the errors specified for the routines fclose and fflush.

SEE ALSO

fclose, fseek

STANDARDS

The **fopen**() and **freopen**() functions conform to ANSI X3.159−1989 (``ANSI C'').

printf, fprintf, sprintf, vprintf, vfprintf, vsprintf − formatted output conversion

SYNOPSIS

```
#include <stdio.h>

int
printf(const char *format, ...)
int
fprintf(FILE *stream, const char *format, ...)
int
sprintf(char *str, const char *format, ...)

#include <stdarg.h>

int
vprintf(const char *format, va_list ap)
int
vfprintf(FILE *stream, const char *format, va_list ap)
int
vsprintf(char *str, const char *format, va_list ap)
```

DESCRIPTION

The **printf**() family of functions produces output according to a *format* as described below. **Printf**() and **vprintf**() write output to *stdout*, the standard output stream; **fprintf**() and **vfprintf**() write output to the given output *stream*; **sprintf**() and **vsprintf**() write to the character string *str*. These functions write the output under the control of a *format* string that specifies how subsequent arguments (or arguments accessed via the variable−length argument facilities of stdarg) are converted for output. These functions return the number of characters printed (not including the trailing `\0' used to end output to strings).

The format string is composed of zero or more directives: ordinary characters (not **%**), which are copied unchanged to the output stream; and conversion specifications, each of which results in fetching zero or more subsequent arguments. Each conversion specification is introduced by the character **%**. The arguments must correspond properly (after type promotion) with the conversion specifier. After the **%**, the following appear in sequence:

- Zero or more of the following flags:
  - A **#** character specifying that the value should be converted to an ``alternate form''. For **c**, **d**, **i**, **n**, **p**, **s**, and **u**, conversions, this option has no effect. For **o** conversions, the precision of the number is increased to force the first character of the output string to a zero (except if a zero value is printed with an explicit precision of zero). For **x** and **X** conversions, a non−zero result has the string `0x' (or `0X' for **X** conversions) prepended to it. For **e**, **E**, **f**, **g**, and **G**, conversions, the result will always contain a decimal point, even if no digits follow it (normally, a decimal point appears in the results of those conversions only if a digit follows). For **g** and **G** conversions, trailing zeros are not removed from the result as they would otherwise be.
  - A zero **`0**' character specifying zero padding. For all conversions except **n**, the converted value is padded on the left with zeros rather than blanks. If a precision is given with a numeric conversion (Mc d, **i**, **o**, **u**, **i**, **x**, and **X**), the **`0**' flag is ignored.
  - A negative field width flag `−' indicates the converted value is to be left adjusted on the field boundary. Except for **n** conversions, the converted value is padded on the

right with blanks, rather than on the left with blanks or zeros. A `−' overrides a **`0'** if both are given.

◊ A space, specifying that a blank should be left before a positive number produced by a signed conversion (**d**, **e**, **E**, **f**, **g**, **G**, or **i**).

◊ A `+' character specifying that a sign always be placed before a number produced by a signed conversion. A `+' overrides a space if both are used.

♦ An optional decimal digit string specifying a minimum field width. If the converted value has fewer characters than the field width, it will be padded with spaces on the left (or right, if the left−adjustment flag has been given) to fill out the field width.

♦ An optional precision, in the form of a period **`.'** followed by an optional digit string. If the digit string is omitted, the precision is taken as zero. This gives the minimum number of digits to appear for **d**, **i**, **o**, **u**, **x**, and **X** conversions, the number of digits to appear after the decimal−point for **e**, **E**, and **f** conversions, the maximum number of significant digits for **g** and **G** conversions, or the maximum number of characters to be printed from a string for **s** conversions.

♦ The optional character **h**, specifying that a following **d**, **i**, **o**, **u**, **x**, or **X** conversion corresponds to a *short int* or *unsigned short int* argument, or that a following **n** conversion corresponds to a pointer to a *short int* argument.

♦ The optional character **l** (ell) specifying that a following **d**, **i**, **o**, **u**, **x**, or **X** conversion applies to a pointer to a *long int* or *unsigned long int* argument, or that a following **n** conversion corresponds to a pointer to a *long int* argument.

♦ The character **L** specifying that a following **e**, **E**, **f**, **g**, or **G** conversion corresponds to a *long double* argument.

♦ A character that specifies the type of conversion to be applied.

A field width or precision, or both, may be indicated by an asterisk `*' instead of a digit string. In this case, an *int* argument supplies the field width or precision. A negative field width is treated as a left adjustment flag followed by a positive field width; a negative precision is treated as though it were missing.

The conversion specifiers and their meanings are:

diouxX

The *int* (or appropriate variant) argument is converted to signed decimal (**d** and **i**), unsigned octal (**o**), unsigned decimal (**u**), or unsigned hexadecimal (**x** and **X**) notation. The letters **abcdef** are used for **x** conversions; the letters **ABCDEF** are used for **X** conversions. The precision, if any, gives the minimum number of digits that must appear; if the converted value requires fewer digits, it is padded on the left with zeros.

eE

The *double* argument is rounded and converted in the style [−]d**.**ddd**e**+−dd where there is one digit before the decimal−point character and the number of digits after it is equal to the precision; if the precision is missing, it is taken as 6; if the precision is zero, no decimal−point character appears. An **E** conversion uses the letter **E** (rather than **e**) to introduce the exponent. The exponent always contains at least two digits; if the value is zero, the exponent is 00.

f

The *double* argument is rounded and converted to decimal notation in the style [−]ddd**.**ddd, where the number of digits after the decimal−point character is equal to the precision specification. If the precision is missing, it is taken as 6; if the precision is explicitly zero, no decimal−point character appears. If a decimal point appears, at least one digit appears before it.

g

The *double* argument is converted in style **f** or **e** (or **E** for **G** conversions). The precision specifies the number of significant digits. If the precision is missing, 6 digits are given; if the precision is zero, it is treated as 1. Style **e** is used if the exponent from its conversion is less than −4 or greater than or equal to the precision. Trailing zeros are removed from the fractional part of the result; a decimal point appears only if it is followed by at least one digit.

c

The *int* argument is converted to an *unsigned char*, and the resulting character is written.

s

The ``*char* *'' argument is expected to be a pointer to an array of character type (pointer to a string). Characters from the array are written up to (but not including) a terminating NUL character; if a precision is specified, no more than the number specified are written. If a precision is given, no null character need be present; if the precision is not specified, or is greater than the size of the array, the array must contain a terminating NUL character.

p

The ``*void* *'' pointer argument is printed in hexadecimal (as if by `%#x' or `%#lx').

n

The number of characters written so far is stored into the integer indicated by the ``*int* *'' (or variant) pointer argument. No argument is converted.

%

A `%' is written. No argument is converted. The complete conversion specification is `%%'.

In no case does a non−existent or small field width cause truncation of a field; if the result of a conversion is wider than the field width, the field is expanded to contain the conversion result.

EXAMPLES

To print a date and time in the form `Sunday, July 3, 10:02', where *weekday* and *month* are pointers to strings:

```
#include <stdio.h>
fprintf(stdout, "%s, %s %d, %.2d:%.2d\n",
        weekday, month, day, hour, min);
```

To print pi to five decimal places:

```
#include <math.h>
#include <stdio.h>
fprintf(stdout, "pi = %.5f\n", 4 * atan(1.0));
```

To allocate a 128 byte string and print into it:

```
#include <stdio.h>
#include <stdlib.h>
#include <stdarg.h>
char *newfmt(const char *fmt, ...)
        {
        char *p;
        va_list ap;
        if ((p = malloc(128)) == NULL)
                return (NULL);
        va_start(ap, fmt);
        (void) vsprintf(p, fmt, ap);
        va_end(ap);
```

```
return (p);
}
```

SEE ALSO

scanf

STANDARDS

The **fprintf**(), **printf**(), **sprintf**(), **vprintf**(), **vfprintf**(), and **vsprintf**() functions conform to ANSI X3.159−1989 (``ANSI C'').

BUGS

Because **sprintf**() and **vsprintf**() assume an infinitely long string, callers must be careful not to overflow the actual space; this is often impossible to assure.

fputc, putc, putchar − output a character to a stream

SYNOPSIS

```
#include <stdio.h>

int
fputc(int c, FILE *stream)
int
putc(int c, FILE *stream)
int
putchar(int c)
```

DESCRIPTION

The **fputc**() function writes the character *c* (converted to an ``unsigned char'') to the output stream pointed to by *stream*.

Putc() acts essentially identically to **fputc**(), but is a macro that expands in−line. It may evaluate *stream* more than once, so arguments given to **putc**() should not be expressions with potential side effects.

Putchar() is identical to **putc**() with an output stream of *stdout*.

RETURN VALUES

The functions, **fputc**(), **putc**() and **putchar**() return the character written. If an error occurs, the value EOF is returned.

SEE ALSO

ferror, fopen, getc, stdio

STANDARDS

The functions **fputc**(), **putc**(), and **putchar**(), conform to ANSI X3.159−1989 (``ANSI C'').

fputs, puts − output a line to a stream

SYNOPSIS

```
#include <stdio.h>

int
fputs(const char *str, FILE *stream)
int
puts(const char *str)
```

DESCRIPTION

The function **fputs**() writes the string pointed to by *str* to the stream pointed to by *stream*.

The function **puts**() writes the string *str*, and a terminating newline character, to the stream *stdout*.

RETURN VALUES

The **fputs**() function returns 0 on success and EOF on error; **puts**() returns a nonnegative integer on success and EOF on error.

ERRORS

[EBADF]
The *stream* supplied is not a writable stream.

SEE ALSO

putc, ferror, stdio

STANDARDS

The functions **fputs**() and **puts**() conform to ANSI X3.159−1989 (``ANSI C'').

fread, fwrite − binary stream input/output

SYNOPSIS

```
#include <stdio.h>

size_t
fread(void *ptr, size_t size, size_t nmemb, FILE *stream)
size_t
fwrite(const void *ptr, size_t size, size_t nmemb, FILE *stream)
```

DESCRIPTION

The function **fread**() reads *nmemb* objects, each *size* bytes long, from the stream pointed to by *stream*, storing them at the location given by *ptr*.

The function **fwrite**() writes *nmemb* objects, each *size* bytes long, to the stream pointed to by *stream*, obtaining them from the location given by *ptr*.

RETURN VALUES

The functions **fread**() and **fwrite**() advance the file position indicator for the stream by the number of bytes read or written. They return the number of objects read or written. If an error occurs, or the end−offile is reached, the return value is a short object count (or zero).

The function **fread**() does not distinguish between end−of−file and error, and callers must use feof and ferror to determine which occurred. The function **fwrite**() returns a value less than *nmemb* only if a write error has occurred.

STANDARDS

The functions **fread**() and **fwrite**() conform to ANSI X3.159−1989 (``ANSI C'').

scanf, fscanf, sscanf − input format conversion

SYNOPSIS

```
#include <stdio.h>

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

DESCRIPTION

The **scanf**() family of functions scans input according to a *format* as described below. This format may contain *conversion specifiers*; the results from such conversions, if any, are stored through the *pointer* arguments. The **scanf**() function reads input from the standard input stream *stdin*, **fscanf**() reads input from the stream pointer *stream*, and **sscanf**() reads its input from the character string pointed to by *str*. Each successive *pointer* argument must correspond properly with each successive conversion specifier (but see `suppression' below). All conversions are introduced by the **%** (percent sign) character. The *format* string may also contain other characters. White space (such as blanks, tabs, or newlines) in the *format* string match any amount of white space, including none, in the input. Everything else matches only itself. Scanning stops when an input character does not match such a format character. Scanning also stops when an input conversion cannot be made (see below).

CONVERSIONS

Following the **%** character introducing a conversion there may be a number of *flag* characters, as follows:

***** Suppresses assignment. The conversion that follows occurs as usual, but no pointer is used; the result of the conversion is simply discarded.

h Indicates that the conversion will be one of **dioux** or **n** and the next pointer is a pointer to a *short int* (rather than *int*).

l Indicates either that the conversion will be one of **dioux** or **n** and the next pointer is a pointer to a *long int* (rather than *int*), or that the conversion will be one of **efg** and the next pointer is a pointer to *double* (rather than *float*).

q Indicates that the conversion will be one of **dioux** or **n** and the next pointer is a pointer to a *quad_t* (rather than *int*).

L Indicates that the conversion will be **efg** and the next pointer is a pointer to *long double*.

In addition to these flags, there may be an optional maximum field width, expressed as a decimal integer, between the **%** and the conversion. If no width is given, a default of `infinity' is used (with one exception, below); otherwise at most this many characters are scanned in processing the conversion. Before conversion begins, most conversions skip white space; this white space is not counted against the field width.

The following conversions are available:

%

Matches a literal `%'. That is, `%%' in the format string matches a single input `%' character. No conversion is done, and assignment does not occur.

d

Matches an optionally signed decimal integer; the next pointer must be a pointer to *int*.

i

Matches an optionally signed integer; the next pointer must be a pointer to *int*. The integer is read in base 16 if it begins with `0x' or `0X', in base 8 if it begins with `0', and in base 10 otherwise. Only characters that correspond to the base are used.

o

Matches an octal integer; the next pointer must be a pointer to *unsigned int*.

u

Matches an optionally signed decimal integer; the next pointer must be a pointer to *unsigned int*.

x

Matches an optionally signed hexadecimal integer; the next pointer must be a pointer to *unsigned int*.

X

Equivalent to **x**.

f

Matches an optionally signed floating−point number; the next pointer must be a pointer to *float*.

e

Equivalent to **f**.

g

Equivalent to **f**.

E

Equivalent to **f**.

G

Equivalent to **f**.

s

Matches a sequence of non−white−space characters; the next pointer must be a pointer to *char*, and the array must be large enough to accept all the sequence and the terminating NUL character. The input string stops at white space or at the maximum field width, whichever occurs first.

c

Matches a sequence of *width* count characters (default 1); the next pointer must be a pointer to *char*, and there must be enough room for all the characters (no terminating NUL is added). The usual skip of leading white space is suppressed. To skip white space first, use an explicit space in the format.

[

Matches a nonempty sequence of characters from the specified set of accepted characters; the next pointer must be a pointer to *char*, and there must be enough room for all the characters in the string, plus a terminating NUL character. The usual skip of leading white space is suppressed. The string is to be made up of characters in (or not in) a particular set; the set is defined by the characters between the open bracket [character and a close bracket] character. The set *excludes* those characters if the first character after the open bracket is a circumflex ^. To include a close bracket in the set, make it the first character after the open bracket or the circumflex; any other position will end the set. The hyphen character − is also special; when placed between two other characters, it adds all intervening characters to the set. To include a hyphen, make it the last character before the final close bracket. For

instance, `[^]0−9−]' means the set `everything except close bracket, zero through nine, and hyphen'. The string ends with the appearance of a character not in the (or, with a circumflex, in) set or when the field width runs out.

p

Matches a pointer value (as printed by `%p' in printf); the next pointer must be a pointer to *void*.

n

Nothing is expected; instead, the number of characters consumed thus far from the input is stored through the next pointer, which must be a pointer to *int*. This is *not* a conversion, although it can be suppressed with the ***** flag.

RETURN VALUES

These functions return the number of input items assigned, which can be fewer than provided for, or even zero, in the event of a matching failure. Zero indicates that, while there was input available, no conversions were assigned; typically this is due to an invalid input character, such as an alphabetic character for a `%d' conversion. The value EOF is returned if an input failure occurs before any conversion such as an endof−file occurs. If an error or end−of−file occurs after conversion has begun, the number of conversions which were successfully completed is returned.

SEE ALSO

strtol, strtoul, strtod, getc, printf

STANDARDS

The functions **fscanf**(), **scanf**(), and **sscanf**() conform to ANSI X3.159−1989 (``ANSI C'').

BUGS

Numerical strings are truncated to 512 characters; for example, **%f** and **%d** are implicitly **%512f** and **%512d**.

perror − write error messages to standard error

SYNOPSIS

```
#include <stdio.h>

void
perror(const char *string)
```

DESCRIPTION

The **perror**() function looks up the language−dependent error message string affiliated with an error number and writes it, followed by a newline, to the standard error stream.

If the argument *string* is non−NULL it is prepended to the message string and separated from it by a colon and a space (`: '). If *string* is NULL only the error message string is printed.

The contents of the error message string is the same as those returned by **strerror**() with argument *errno*.

SEE ALSO

strerror

STANDARDS

The **perror**() function conforms to ANSI X3.159−1989 (``ANSI C'').

remove − remove directory entry

SYNOPSIS

```
#include <stdio.h>

int
remove(const char *path)
```

DESCRIPTION

The **remove**() function deletes the file referenced by *path*.

RETURN VALUES

Upon successful completion, **remove**() returns 0. Otherwise, −1 is returned and the global variable *errno* is set to indicate the error.

STANDARDS

The **remove**() function conforms to ANSI X3.159−1989 (``ANSI C").

setbuf, setvbuf − stream buffering operations

SYNOPSIS

```
#include <stdio.h>

void
setbuf(FILE *stream, char *buf)
int
setvbuf(FILE *stream, char *buf, int mode, size_t size)
```

DESCRIPTION

The three types of buffering available are unbuffered, block buffered, and line buffered. When an output stream is unbuffered, information appears on the destination file or terminal as soon as written; when it is block buffered many characters are saved up and written as a block; when it is line buffered characters are saved up until a newline is output or input is read from any stream attached to a terminal device (typically stdin). The function fflush may be used to force the block out early. (See fclose.) Normally all files are block buffered. When the first I/O operation occurs on a file, malloc is called, and an optimally−sized buffer is obtained. If a stream refers to a terminal (as *stdout* normally does) it is line buffered. The standard error stream *stderr* is always unbuffered.

The **setvbuf**() function may be used to alter the buffering behavior of a stream. The *mode* parameter must be one of the following three macros:

```
_IONBF  unbuffered

_IOLBF  line buffered

_IOFBF  fully buffered
```

The *size* parameter may be given as zero to obtain deferred optimal−size buffer allocation as usual. If it is not zero, then except for unbuffered files, the *buf* argument should point to a buffer at least *size* bytes long; this buffer will be used instead of the current buffer. (If the *size* argument is not zero but *buf* is NULL, a buffer of the given size will be allocated immediately, and released on close. This is an extension to ANSI C; portable code should use a size of 0 with any NULL buffer.) The **setvbuf**() function may be used at any time, but may have peculiar side effects (such as discarding input or flushing output) if the stream is ``active''. Portable applications should call it only once on any given stream, and before any I/O is performed.

The other three calls are, in effect, simply aliases for calls to **setvbuf**(). Except for the lack of a return value, the **setbuf**() function is exactly equivalent to the call

```
setvbuf(stream, buf, buf ? _IOFBF : _IONBF, BUFSIZ);
```

RETURN VALUES

The **setvbuf**() function returns 0 on success, or EOF if the request cannot be honored (note that the stream is still functional in this case).

SEE ALSO

fopen, fclose, fread, malloc, puts, printf

STANDARDS

The **setbuf**() and **setvbuf**() functions conform to ANSI X3.159−1989 (``ANSI C'').

tmpfile, tmpnam − temporary file routines

SYNOPSIS

```
#include <stdio.h>

FILE *
tmpfile(void)
char *
tmpnam(char *str)
```

DESCRIPTION

The **tmpfile**() and **tmpnam**() functions are not implemented.

The **tmpfile**() function creates a temporary binary file that will be automatically deleted when the last reference to it is closed. The file is opened with the access value `wb+'.

The **tmpnam**() function returns a pointer to a file name, which is not the name of an existing file. If the argument *s* is non−NULL, the file name is copied to the buffer it references. Otherwise, the file name is copied to a static buffer. In either case, **tmpnam**() returns a pointer to the file name.

The buffer referenced by *s* is expected to be at least L_tmpnam bytes in length. L_tmpnam is defined in the include file <*stdio.h*>.

RETURN VALUES

The **tmpfile**() function returns a pointer to an open file stream on success, and a NULL pointer on error.

The **tmpnam**() function returns a pointer to a file name on success, and a NULL pointer on error.

ERRORS

The **tmpfile**() function may fail and set the global variable *errno* for any of the errors specified for the library functions fdopen.

STANDARDS

The **tmpfile**() and **tmpnam**() functions conform to ANSI X3.159−1989 (``ANSI C'').

ungetc − un−get character from input stream

SYNOPSIS

```
#include <stdio.h>

int
ungetc(int c, FILE *stream)
```

DESCRIPTION

The **ungetc**() function pushes the character *c* (converted to an unsigned char) back onto the input stream pointed to by *stream*. The pushed−backed characters will be returned by subsequent reads on the stream (in reverse order). A successful intervening call, using the same stream, to one of the file positioning functions (fseek, fsetpos, or rewind) will discard the pushed back characters.

One character of push−back is guaranteed, but as long as there is sufficient memory, an effectively infinite amount of pushback is allowed.

If a character is successfully pushed−back, the end−of−file indicator for the stream is cleared.

RETURN VALUES

The **ungetc**() function returns the character pushed−back after the conversion, or EOF if the operation fails. If the value of the argument *c* character equals EOF, the operation will fail and the stream will remain unchanged.

SEE ALSO

getc, fseek, setvbuf

STANDARDS

The **ungetc**() function conforms to ANSI X3.159−1989 (``ANSI C'').

stdlib − general utilities

SYNOPSIS

#include <***stdlib.h***>

void **abort**(*void*)**;**
Abnormal program termination.
int ***abs****(int);*
Compute an absolute value.
int ***atexit****(void (*)(void);*
Register a function to be called on program exit.
double ***atof****(const char *);*
Convert ASCII string to double.
int ***atoi****(const char *);*
Convert ASCII string to integer.
long ***atol****(const char *);*
Convert ASCII string to long.
void ****bsearch****(const void *, const void *, size_t, size_t, int (*)(const void *, const void *));*
Binary search of a sorted table.
void ****calloc****(size_t, size_t);*
Allocate zero initialized memory.
div_t ***div****(int, int);*
Return quotient and remainder from division.
void ***exit****(int);*
Perform normal program termination.
void ***free****(void *);*
Free allocated memory.
char ****getenv****(const char *);*
Get an environment variable.
long ***labs****(long);*
Compute an absolute value of a long integer.
ldiv_t ***ldiv****(long, long);*
Return a quotient an remainder from a long division.
void ****malloc****(size_t);*
Allocate memory.
void ***qsort****(void *, size_t, size_t, int (*)(const void *, const void *));*
Sort function.
void ***rand****(void);*
Return a pseudo random number.
void ****realloc****(void *, size_t);*
Reallocate allocated memory with a new size.
void ***srand****(unsigned);*
Set the pseudo random number seed.
double ***strtod****(const char *, char **);*
Convert a string to a double.
long ***strtol****(const char *, char **);*
Convert a string to a long integer.
unsigned long ***strtoul****(const char *, char **);*
Convert a string to an unsigned long integer.
int ***system****(const char *);*

Pass a command to the shell.

abort − cause abnormal program termination

SYNOPSIS

```
#include <stdlib.h>

void
abort(void)
```

DESCRIPTION

The **abort**() function causes abnormal program termination to occur.

RETURN VALUES

The **abort** function never returns.

SEE ALSO

exit

STANDARDS

The **abort**() function conforms to ANSI X3.159−1989 (``ANSI C'').

abs − integer absolute value function

SYNOPSIS

```
#include <stdlib.h>

int
abs(int j)
```

DESCRIPTION

The **abs**() function computes the absolute value of the integer *j*.

RETURN VALUES

The **abs**() function returns the absolute value.

SEE ALSO

floor, labs

STANDARDS

The **abs**() function conforms to ANSI X3.159−1989 (``ANSI C'').

BUGS

The absolute value of the most negative integer remains negative.

atexit − register a function to be called on exit

SYNOPSIS

```
#include <stdlib.h>

int
atexit(void (*function)(void))
```

DESCRIPTION

The **atexit**() function registers the given *function* to be called at program exit, whether via exit or via return from the program's *main*. Functions so registered are called in reverse order; no arguments are passed. At least 32 functions can always be registered, and more are allowed as long as sufficient memory can be allocated.

RETURN VALUES

The **atexit**() function returns the value 0 if successful; otherwise the value −1 is returned and the global variable *errno* is set to indicate the error.

ERRORS

[ENOMEM]
No memory was available to add the function to the list. The existing list of functions is unmodified.

SEE ALSO

exit

STANDARDS

The **atexit**() function conforms to ANSI X3.159−1989 (``ANSI C").

atof − convert ASCII string to double

SYNOPSIS

```
#include <stdlib.h>

double
atof(const char *nptr)
```

DESCRIPTION

The **atof**() function converts the initial portion of the string pointed to by *nptr* to *double* representation.

It is equivalent to:

```
strtod(nptr, (char **)NULL);
```

SEE ALSO

atoi, atol, strtod, strtol, strtoul

STANDARDS

The **atof**() function conforms to ANSI X3.159−1989 (``ANSI C'').

atoi − convert ASCII string to integer

SYNOPSIS

```
#include <stdlib.h>

int
atoi(const char *nptr)
```

DESCRIPTION

The **atoi**() function converts the initial portion of the string pointed to by *nptr* to *integer* representation.

It is equivalent to:

```
(int)strtol(nptr, (char **)NULL, 10);
```

SEE ALSO

atof, atol, strtod, strtol, strtoul

STANDARDS

The **atoi**() function conforms to ANSI X3.159−1989 (``ANSI C'').

atol − convert ASCII string to long integer

SYNOPSIS

```
#include <stdlib.h>

long
atol(const char *nptr)
```

DESCRIPTION

The **atol**() function converts the initial portion of the string pointed to by *nptr* to *long integer* representation.

It is equivalent to:

```
strtol(nptr, (char **)NULL, 10);
```

SEE ALSO

atof, atoi, strtod, strtol, strtoul

STANDARDS

The **atol**() function conforms to ANSI X3.159−1989 (``ANSI C'').

bsearch − binary search of a sorted table

SYNOPSIS

```
#include <stdlib.h>

void *
bsearch(const void *key, const void *base, size_t nmemb, size_t size,
        int (*compar) (const void *, const void *))
```

DESCRIPTION

The **bsearch**() function searches an array of *nmemb* objects, the initial member of which is pointed to by *base*, for a member that matches the object pointed to by *key*. The size of each member of the array is specified by *size*.

The contents of the array should be in ascending sorted order according to the comparison function referenced by *compar*. The *compar* routine is expected to have two arguments which point to the *key* object and to an array member, in that order, and should return an integer less than, equal to, or greater than zero if the *key* object is found, respectively, to be less than, to match, or be greater than the array member.

RETURN VALUES

The **bsearch**() function returns a pointer to a matching member of the array, or a null pointer if no match is found. If two members compare as equal, which member is matched is unspecified.

SEE ALSO

qsort

STANDARDS

The **bsearch**() function conforms to ANSI X3.159−1989 (``ANSI C'').

calloc − allocate clean memory (zero initialized space)

SYNOPSIS

```
#include <stdlib.h>

void *
calloc(size_t nmemb, size_t size)
```

DESCRIPTION

The **calloc**() function allocates space for an array of *nmemb* objects, each of whose size is *size*. The space is initialized to all bits zero.

RETURN VALUES

The **calloc**() function returns a pointer to the allocated space if successful; otherwise a null pointer is returned.

SEE ALSO

malloc, realloc, free

STANDARDS

The **calloc**() function conforms to ANSI X3.159−1989 (``ANSI C'').

div − return quotient and remainder from division

SYNOPSIS

```
#include <stdlib.h>

div_t
div(int num, int denom)
```

DESCRIPTION

The **div**() function computes the value *num/denom* and returns the quotient and remainder in a structure named *div_t* that contains two *int* members named *quot* and *rem*.

SEE ALSO

ldiv

STANDARDS

The **div**() function conforms to ANSI X3.159−1989 (``ANSI C'').

exit − perform normal program termination

SYNOPSIS

```
#include <stdlib.h>

void
exit(int status)
```

DESCRIPTION

Exit() terminates a process.

Before termination it performs the following functions in the order listed:

1. Call the functions registered with the atexit function, in

   the reverse order of their registration.
2. Flush all open output streams.
3. Close all open streams.

RETURN VALUES

The **exit**() function never returns.

SEE ALSO

atexit

STANDARDS

The **exit**() function conforms to ANSI X3.159−1989 (``ANSI C'').

free − free up memory allocated with malloc, calloc or realloc

SYNOPSIS

```
#include <stdlib.h>

void
free(void *ptr)
```

DESCRIPTION

The **free**() function causes the space pointed to by *ptr* to be deallocated, that is, made available for further allocation. If *ptr* is a null pointer, no action occurs. Otherwise, if the argument does not match a pointer earlier returned by the calloc, malloc, or realloc function, or if the space has been deallocated by a call to **free**() or realloc, general havoc may occur.

RETURN VALUES

The **free**() function returns no value.

SEE ALSO

calloc, malloc, realloc

STANDARDS

The **free**() function conforms to ANSI X3.159−1989 (``ANSI C'').

getenv − get environment variable

SYNOPSIS

```
#include <stdlib.h>

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

DESCRIPTION

The **getenv**() function obtains the current value of the environment variable, *name*. If the variable *name* is not in the current environment , a null pointer is returned.

STANDARDS

The **getenv**() function conforms to ANSI X3.159−1989 (``ANSI C'').

labs − return the absolute value of a long integer

SYNOPSIS

```
#include <stdlib.h>

long
labs(long j)
```

DESCRIPTION

The **labs**() function returns the absolute value of the long integer *j*.

SEE ALSO

abs, floor, math

STANDARDS

The **labs**() function conforms to ANSI X3.159−1989 (``ANSI C'').

BUGS

The absolute value of the most negative integer remains negative.

ldiv − return quotient and remainder from division

SYNOPSIS

```
#include <stdlib.h>

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

DESCRIPTION

The **ldiv**() function computes the value *num/denom* and returns the quotient and remainder in a structure named *ldiv_t* that contains two *long integer* members named *quot* and *rem*.

SEE ALSO

div, math

STANDARDS

The **ldiv**() function conforms to ANSI X3.159−1989 (``ANSI C'').

malloc − general memory allocation function

SYNOPSIS

```
#include <stdlib.h>

void *
malloc(size_t size)
```

DESCRIPTION

The **malloc**() function allocates uninitialized space for an object whose size is specified by *size*. The **malloc**() function maintains multiple lists of free blocks according to size, allocating space from the appropriate list.

The allocated space is suitably aligned (after possible pointer coercion) for storage of any type of object.

RETURN VALUES

The **malloc**() function returns a pointer to the allocated space if successful; otherwise a null pointer is returned.

SEE ALSO

free, calloc, realloc

STANDARDS

The **malloc**() function conforms to ANSI X3.159−1989 (``ANSI C'').

qsort − sort function

SYNOPSIS

```
#include <stdlib.h>

void
qsort(void *base, size_t nmemb, size_t size,
        int (*compar)(const void *, const void *))
```

DESCRIPTION

The **qsort**() function is a modified partition−exchange sort, or quicksort.

The **qsort**() function sorts an array of *nmemb* objects, the initial member of which is pointed to by *base*. The size of each object is specified by *size*.

The contents of the array *base* are sorted in ascending order according to a comparison function pointed to by *compar*, which requires two arguments pointing to the objects being compared.

The comparison function must return an integer less than, equal to, or greater than zero if the first argument is considered to be respectively less than, equal to, or greater than the second.

The functions **qsort**() is *not* stable, that is, if two members compare as equal, their order in the sorted array is undefined.

RETURN VALUES

The **qsort**() function returns no value.

STANDARDS

The **qsort**() function conforms to ANSI X3.159−1989 (``ANSI C'').

rand, srand − random number generator

SYNOPSIS

```
#include <stdlib.h>

void
srand(unsigned seed)
int
rand(void)
```

DESCRIPTION

The **rand**() function computes a sequence of pseudo−random integers in the range of 0 to RAND_MAX (as defined by the header file <*stdlib.h*>).

The **srand**() function sets its argument as the seed for a new sequence of pseudo−random numbers to be returned by **rand**(). These sequences are repeatable by calling **srand**() with the same seed value.

If no seed value is provided, the functions are automatically seeded with a value of 1.

STANDARDS

The **rand**() and **srand**() functions conform to ANSI X3.159−1989 (``ANSI C'').

realloc − reallocation of memory function

SYNOPSIS

```
#include <stdlib.h>

void *
realloc(void *ptr, size_t size)
```

DESCRIPTION

The **realloc**() function changes the size of the object pointed to by *ptr* to the size specified by *size*. The contents of the object are unchanged up to the lesser of the new and old sizes. If the new size is larger, the value of the newly allocated portion of the object is indeterminate. If *ptr* is a null pointer, the **realloc**() function behaves like the malloc function for the specified size. Otherwise, if *ptr* does not match a pointer earlier returned by the calloc, malloc, or **realloc**() function, or if the space has been deallocated by a call to the free or **realloc**() function, unpredictable and usually detrimental behavior will occur. If the space cannot be allocated, the object pointed to by *ptr* is unchanged. If *size* is zero and *ptr* is not a null pointer, the object it points to is freed.

The **realloc**() function returns either a null pointer or a pointer to the possibly moved allocated space.

SEE ALSO

calloc, free, malloc,

STANDARDS

The **realloc**() function conforms to ANSI X3.159−1989 (``ANSI C'').

strtod − convert string to double

SYNOPSIS

```
#include <stdlib.h>

double
strtod(const char *nptr, char **endptr)
```

DESCRIPTION

The **strtod**() function converts the initial portion of the string pointed to by *nptr* to *double* representation.

The expected form of the string is an optional plus (``+") or minus sign (``−") followed by a sequence of digits optionally containing a decimalpoint character, optionally followed by an exponent. An exponent consists of an ``E" or ``e", followed by an optional plus or minus sign, followed by a sequence of digits.

Leading white−space characters in the string (as defined by the isspace function) are skipped.

RETURN VALUES

The **strtod**() function returns the converted value, if any.

If *endptr* is not NULL, a pointer to the character after the last character used in the conversion is stored in the location referenced by *endptr*.

If no conversion is performed, zero is returned and the value of *nptr* is stored in the location referenced by *endptr*.

If the correct value would cause overflow, plus or minus HUGE_VAL is returned (according to the sign of the value), and ERANGE is stored in *errno*. If the correct value would cause underflow, zero is returned and ERANGE is stored in *errno*.

ERRORS

[ERANGE]
Overflow or underflow occurred.

SEE ALSO

atof, atoi, atol, strtol, strtoul

STANDARDS

The **strtod**() function conforms to ANSI X3.159−1989 (``ANSI C").

strtol − convert string value to a long integer

SYNOPSIS

```
#include <stdlib.h>

long
strtol(const char *nptr, char **endptr, int base)
```

DESCRIPTION

The **strtol**() function converts the string in *nptr* to a *long* value. The conversion is done according to the given *base*, which must be between 2 and 36 inclusive, or be the special value 0.

The string may begin with an arbitrary amount of white space (as determined by isspace) followed by a single optional `+' or `−' sign. If *base* is zero or 16, the string may then include a `0x' prefix, and the number will be read in base 16; otherwise, a zero *base* is taken as 10 (decimal) unless the next character is `0', in which case it is taken as 8 (octal).

The remainder of the string is converted to a *long* value in the obvious manner, stopping at the first character which is not a valid digit in the given base. (In bases above 10, the letter `A' in either upper or lower case represents 10, `B' represents 11, and so forth, with `Z' representing 35.) If *endptr* is non nil, **strtol**() stores the address of the first invalid character in **endptr*. If there were no digits at all, however, **strtol**() stores the original value of *nptr* in **endptr*. (Thus, if **nptr* is not `\0' but ***endptr* is `\0' on return, the entire string was valid.)

RETURN VALUES

The **strtol**() function returns the result of the conversion, unless the value would underflow or overflow. If an underflow occurs, **strtol**() returns LONG_MIN. If an overflow occurs, **strtol**() returns LONG_MAX. In both cases, *errno* is set to ERANGE.

ERRORS

[ERANGE]
The given string was out of range; the value converted has been clamped.

SEE ALSO

atof, atoi, atol, strtod, strtoul

STANDARDS

The **strtol**() function conforms to ANSI X3.159−1989 (``ANSI C'').

strtoul − convert a string to an unsigned long integer

SYNOPSIS

```
#include <stdlib.h>

unsigned long
strtoul(const char *nptr, char **endptr, int base)
```

DESCRIPTION

The **strtoul**() function converts the string in *nptr* to an *unsigned long* value. The conversion is done according to the given *base*, which must be between 2 and 36 inclusive, or be the special value 0.

The string may begin with an arbitrary amount of white space (as determined by isspace) followed by a single optional `+' or `−' sign. If *base* is zero or 16, the string may then include a `0x' prefix, and the number will be read in base 16; otherwise, a zero *base* is taken as 10 (decimal) unless the next character is `0', in which case it is taken as 8 (octal).

The remainder of the string is converted to an *unsigned long* value in the obvious manner, stopping at the end of the string or at the first character that does not produce a valid digit in the given base. (In bases above 10, the letter `A' in either upper or lower case represents 10, `B' represents 11, and so forth, with `Z' representing 35.) If *endptr* is non nil, **strtoul**() stores the address of the first invalid character in **endptr*. If there were no digits at all, however, **strtoul**() stores the original value of *nptr* in **endptr*. (Thus, if **nptr* is not `\0' but ***endptr* is `\0' on return, the entire string was valid.)

RETURN VALUES

The **strtoul**() function returns either the result of the conversion or, if there was a leading minus sign, the negation of the result of the conversion, unless the original (non−negated) value would overflow; in the latter case, **strtoul**() returns ULONG_MAX and sets the global variable *errno* to ERANGE.

ERRORS

[ERANGE]
The given string was out of range; the value converted has been clamped.

SEE ALSO

strtol

STANDARDS

The **strtoul**() function conforms to ANSI X3.159−1989 (``ANSI C'').

system − pass a command to the shell

SYNOPSIS

```
#include <stdlib.h>

int
system(const char *string)
```

DESCRIPTION

The **system**() function is not implemented.

RETURN VALUES

Returns 0 (Command processor not present).

STANDARDS

The **system**() function conforms to ANSI X3.159−1989 (``ANSI C'') and .

string − string specific functions

SYNOPSIS

***#include** <**string.h**>*
*void **memchr**(const void *, int, size_t);*
Locate a byte in a byte string.
*int **memcmp**(const void *, const void *, size_t);*
Compare byte strings.
*void ***memcpy**(void *, const void *, size_t);*
Copy a byte string.
*void ***memmove**(void *, const void *, size_t);*
Copy an overlapping byte string.
*void ***memset**(void *, int, size_t);*
Write a byte to a byte string.
*char ***strcat**(char *s, const char * append);*
Concatenate two strings.
*char ***strncat**(char *, const char *, size_t);*
Concatenate two strings with a maximum size.
*char ***strchr**(const char *, int);*
Locate the first occurance of a character in a string.
*char ***strrchr**(const char *, int);*
Locate the last occurance of a character in a string.
*int **strcmp**(const char *, const char *);*
Compare two strings.
*int **strncmp**(const char *, const char *, size_t);*
Compare two strings with a maximum size.
*int **strcoll**(const char *, const char *);*
Compare two strings with current collation.
*char ***strcpy**(char *, const char *);*
Copy a string.
*char ***strncpy**(char *, const char *, size_t);*
Copy a string with a maximum size.
*char ***strerror**(int);*
Get the error message string.
*size_t **strlen**(const char *);*
Find the length of a string.
*char ***strpbrk**(const char *, const char *);*
Locate multiple characters in a string.
*size_t **strspn**(const char *, const char *);*
Span a string.
*size_t **strcspn**(const char *, const char *);*
Span the complement of a string.
*char ***strstr**(const char *, const char *);*
Locate a substring in a string.
*char ***strtok**(char *, const char *);*
Find tokens in a string.
*size_t **strxfrm**(char *, const char *,size_t);*
Transform a string under locale.

DESCRIPTION

The string functions functions manipulate strings terminated by a null byte.

See the specific manual pages for more information.

Except as noted in their specific manual pages, the string functions do not test the destination for size limitations.

STANDARDS

The **memchr**(), **memcmp**(), **memcpy**(), **memmove**(), **memset**(), **strcat**(), **strncat**(), **strchr**(), **strrchr**(), **strcmp**(), **strncmp**(), **strcpy**(), **strncpy**(), **strerror**(), **strlen**(), **strpbrk**(), **strspn**(), **strcspn**(), **strstr**(), **strtok**(), and **strxfrm**() functions conform to ANSI X3.159−1989 (``ANSI C'').

memchr − locate byte in byte string

SYNOPSIS

```
#include <string.h>

void *
memchr(const void *b, int c, size_t len)
```

DESCRIPTION

The **memchr**() function locates the first occurrence of *c* (converted to an unsigned char) in string *b*.

RETURN VALUES

The **memchr**() function returns a pointer to the byte located, or NULL if no such byte exists within *len* bytes.

SEE ALSO

strchr, strcspn, strpbrk, strrchr, strspn, strstr, strtok

STANDARDS

The **memchr**() function conforms to ANSI X3.159−1989 (``ANSI C'').

memcmp − compare byte string

SYNOPSIS

```
#include <string.h>

int
memcmp(const void *b1, const void *b2, size_t len)
```

DESCRIPTION

The **memcmp**() function compares byte string *b1* against byte string *b2*. Both strings are assumed to be *len* bytes long.

RETURN VALUES

The **memcmp**() function returns zero if the two strings are identical, otherwise returns the difference between the first two differing bytes (treated as unsigned char values, so that `\200' is greater than `\0', for example). Zero−length strings are always identical.

SEE ALSO

strcmp, strcoll, strxfrm

STANDARDS

The **memcmp**() function conforms to ANSI X3.159−1989 (``ANSI C'').

memcpy − copy byte string

SYNOPSIS

```
#include <string.h>

void *
memcpy(void *dst, const void *src, size_t len)
```

DESCRIPTION

The **memcpy**() function copies *len* bytes from string *src* to string *dst*.

RETURN VALUES

The **memcpy**() function returns the original value of *dst*.

SEE ALSO

memmove, strcpy

STANDARDS

The **memcpy**() function conforms to ANSI X3.159−1989 (``ANSI C'').

memmove − copy overlapping byte string

SYNOPSIS

```
#include <string.h>

void *
memmove(void *dst, const void *src, size_t len)
```

DESCRIPTION

The **memmove**() function copies *len* bytes from string *src* to string *dst*. The two strings may overlap; the copy is always done in a non−destructive manner.

RETURN VALUES

The **memmove**() function returns the original value of *dst*.

SEE ALSO

memcpy, strcpy

STANDARDS

The **memmove**() function conforms to ANSI X3.159−1989 (``ANSI C'').

memset − write a byte to byte string

SYNOPSIS

```
#include <string.h>

void *
memset(void *b, int c, size_t len)
```

DESCRIPTION

The **memset**() function writes *len* bytes of value *c* (converted to an unsigned char) to the string *b*.

STANDARDS

The **memset**() function conforms to ANSI X3.159−1989 (``ANSI C'').

strcat − concatenate strings

SYNOPSIS

```
#include <string.h>

char *
strcat(char *s, const char *append)
char *
strncat(char *s, const char *append, size_t count)
```

DESCRIPTION

The **strcat**() and **strncat**() functions append a copy of the null−terminated string *append* to the end of the null−terminated string *s*, then add a terminating `\0'. The string *s* must have sufficient space to hold the result.

The **strncat**() function appends not more than *count* characters.

RETURN VALUES

The **strcat**() and **strncat**() functions return the pointer *s*.

SEE ALSO

memcpy, memmove, strcpy

STANDARDS

The **strcat**() and **strncat**() functions conform to ANSI X3.159−1989 (``ANSI C'').

strchr − locate character in string

SYNOPSIS

```
#include <string.h>

char *
strchr(const char *s, int c)
```

DESCRIPTION

The **strchr**() function locates the first occurrence of *c* in the string pointed to by *s*. The terminating NULL character is considered part of the string. If *c* is `\0', **strchr**() locates the terminating `\0'.

RETURN VALUES

The function **strchr**() returns a pointer to the located character, or NULL if the character does not appear in the string.

SEE ALSO

memchr, strcspn, strpbrk, strrchr, strspn, strstr, strtok

STANDARDS

The **strchr**() function conforms to ANSI X3.159−1989 (``ANSI C'').

strcmp − compare strings

SYNOPSIS

```
#include <string.h>

int
strcmp(const char *s1, const char *s2)
int
strncmp(const char *s1, const char *s2, size_t len)
```

DESCRIPTION

The **strcmp**() and **strncmp**() functions lexicographically compare the nullterminated strings *s1* and *s2*.

RETURN VALUES

The **strcmp**() and **strncmp**() return an integer greater than, equal to, or less than 0, according as the string *s1* is greater than, equal to, or less than the string *s2*. The comparison is done using unsigned characters, so that `\200' is greater than `\0'.

The **strncmp**() compares not more than *len* characters.

SEE ALSO

memcmp, strcoll, strxfrm

STANDARDS

The **strcmp**() and **strncmp**() functions conform to ANSI X3.159−1989 (``ANSI C'').

strcoll − compare strings according to current collation

SYNOPSIS

```
#include <string.h>

int
strcoll(const char *s1, const char *s2)
```

DESCRIPTION

The **strcoll**() function lexicographically compares the null−terminated strings *s1* and *s2* according to the current locale collation and returns an integer greater than, equal to, or less than 0, according as *s1* is greater than, equal to, or less than *s2*.

SEE ALSO

memcmp, strcmp, strxfrm

STANDARDS

The **strcoll**() function conforms to ANSI X3.159−1989 (``ANSI C'').

strcpy − copy strings

SYNOPSIS

```
#include <string.h>

char *
strcpy(char *dst, const char *src)
char *
strncpy(char *dst, const char *src, size_t len)
```

DESCRIPTION

The **strcpy**() and **strncpy**() functions copy the string *src* to *dst* (including the terminating `\0' character).

The **strncpy**() copies not more than *len* characters into *dst*, appending `\0' characters if *src* is less than *len* characters long, and *not* terminating *dst* if *src* is more than *len* characters long.

RETURN VALUES

The **strcpy**() and **strncpy**() functions return *dst*.

EXAMPLES

The following sets ``chararray'' to ``abc\0\0\0'':

```
(void)strncpy(chararray, "abc", 6).
```

The following sets ``chararray'' to ``abcdef'':

```
(void)strncpy(chararray, "abcdefgh", 6);
```

SEE ALSO

memcpy, memmove

STANDARDS

The **strcpy**() and **strncpy**() functions conform to ANSI X3.159−1989 (``ANSI C'').

strcspn − span the complement of a string

SYNOPSIS

```
#include <string.h>

size_t
strcspn(const char *s, const char *charset)
```

DESCRIPTION

The **strcspn**() function spans the initial part of the null−terminated string *s* as long as the characters from *s* do not occur in string *charset* (it spans the *complement* of *charset*).

RETURN VALUES

The **strcspn**() function returns the number of characters spanned.

SEE ALSO

memchr, strchr, strpbrk, strrchr, strspn, strstr, strtok

STANDARDS

The **strcspn**() function conforms to ANSI X3.159−1989 (``ANSI C'').

strerror − get error message string

SYNOPSIS

```
#include <string.h>

char *
strerror(int errnum)
```

DESCRIPTION

The **strerror**() function returns a pointer to the language−dependent error message string affiliated with an error number.

The array pointed to is not to be modified by the program, but may be overwritten by subsequent calls to **strerror**().

SEE ALSO

perror

STANDARDS

The **strerror**() function conforms to ANSI X3.159−1989 (``ANSI C'').

strlen − find length of string

SYNOPSIS

```
#include <string.h>

size_t
strlen(const char *s)
```

DESCRIPTION

The **strlen**() function computes the length of the string *s*.

RETURN VALUES

The **strlen**() function returns the number of characters that precede the terminating NUL character.

SEE ALSO

string

STANDARDS

The **strlen**() function conforms to ANSI X3.159−1989 (``ANSI C'').

strpbrk − locate multiple characters in string

SYNOPSIS

```
#include <string.h>

char *
strpbrk(const char *s, const char *charset)
```

DESCRIPTION

The **strpbrk**() function locates in the null−terminated string *s* the first occurrence of any character in the string *charset* and returns a pointer to this character. If no characters from *charset* occur anywhere in *s* **strpbrk**() returns NULL.

SEE ALSO

memchr, strchr, strcspn, strrchr, strspn, strstr, strtok

STANDARDS

The **strpbrk**() function conforms to ANSI X3.159−1989 (``ANSI C'').

strrchr − locate character in string

SYNOPSIS

```
#include <string.h>

char *
strrchr(const char *s, int c)
```

DESCRIPTION

The **strrchr**() function locates the last occurrence of *c* (converted to a char) in the string *s*. If *c* is `\0', **strrchr**() locates the terminating `\0'.

RETURN VALUES

The **strrchr**() function returns a pointer to the character, or a null pointer if *c* does not occur anywhere in *s*.

SEE ALSO

memchr, strchr, strcspn, strpbrk, strspn, strstr, strtok

STANDARDS

The **strrchr**() function conforms to ANSI X3.159−1989 (``ANSI C'').

strspn − span a string

SYNOPSIS

```
#include <string.h>

size_t
strspn(const char *s, const char *charset)
```

DESCRIPTION

The **strspn**() function spans the initial part of the null−terminated string *s* as long as the characters from *s* occur in string *charset*.

RETURN VALUES

The **strspn**() function returns the number of characters spanned.

SEE ALSO

memchr, strchr, strcspn, strpbrk, strrchr, strstr, strtok

STANDARDS

The **strspn**() function conforms to ANSI X3.159−1989 (``ANSI C'').

strstr − locate a substring in a string

SYNOPSIS

```
#include <string.h>

char *
strstr(const char *big, const char *little)
```

DESCRIPTION

The **strstr**() function locates the first occurrence of the null−terminated string *little* in the null−terminated string *big*. If *little* is the empty string, **strstr**() returns *big*; if *little* occurs nowhere in *big*, **strstr**() returns NULL; otherwise **strstr**() returns a pointer to the first character of the first occurrence of *little*.

SEE ALSO

memchr, strchr, strcspn, strpbrk, strrchr, strspn, strtok

STANDARDS

The **strstr**() function conforms to ANSI X3.159−1989 (``ANSI C'').

strtok − string token operation

SYNOPSIS

```
#include <string.h>

char *
strtok(char *str, const char *sep)
```

DESCRIPTION

The **strtok**() function is used to isolate sequential tokens in a null−terminated string, *str*. These tokens are separated in the string by at least one of the characters in *sep*. The first time that **strtok**() is called, *str* should be specified; subsequent calls, wishing to obtain further tokens from the same string, should pass a null pointer instead. The separator string, *sep*, must be supplied each time, and may change between calls.

The **strtok**() function returns a pointer to the beginning of each subsequent token in the string, after replacing the separator character itself with a NUL character. When no more tokens remain, a null pointer is returned.

SEE ALSO

memchr, strchr, strcspn, strpbrk, strrchr, strspn, strstr

STANDARDS

The **strtok**() function conforms to ANSI X3.159−1989 (``ANSI C'').

BUGS

There is no way to get tokens from multiple strings simultaneously. **strok**() is not reentrant.

strxfrm − transform a string under locale

SYNOPSIS

```
#include <string.h>

size_t
strxfrm(char *dst, const char *src, size_t n)
```

DESCRIPTION

The **strxfrm**() function does something horrible (see ANSI standard). In this implementation it just copies.

SEE ALSO

memcmp, strcmp, strcoll

STANDARDS

The **strxfrm**() function conforms to ANSI X3.159−1989 (``ANSI C'').

time − date and time handling

SYNOPSIS

#include <***time.h***>

double **asctime**(*double*)**;**
Convert time to ASCII.
clock_t ***clock****(void);*
Determine processor time used.
char ****ctime****(const time_t *);*
Convert a *time_t* to an ASCII time.
double ***difftime****(time_t, time_t);*
Compute the number of seconds between to times.
struct tm ****gmtime****(const time_t);*
Convert a *time_t* to Coordinated Universal Time.
struct tm ****localtime****(const time_t);*
Convert a *time_t* to local time.
time_t ***mktime****(struct tm *);*
Convert local time back into a *time_t*.
size_t ***strftime****(char *, size_t, const char *, const struct tm *);*
Format date and time.
time_t ***time****(time_t *);*
Get time of day.

asctime, ctime, difftime, gmtime, localtime, mktime − date and time to ASCII

SYNOPSIS

```
#include <time.h>

char *ctime(const time_t *clock)

double difftime(time_t time1, time_t time0)

char *asctime(const struct tm *tm)

struct tm *localtime(const time_t *clock)

struct tm *gmtime(const time_t *clock)

time_t mktime(struct tm *tm)
```

DESCRIPTION

Ctime converts a long integer, pointed to by *clock*, representing the time in seconds since 00:00:00 UTC, January 1, 1970, and returns a pointer to a 26−character string of the form

```
Thu Nov 24 18:22:48 1986\n\0
```

All the fields have constant width.

Localtime and *gmtime* return pointers to ``tm" structures, described below. *Localtime* corrects for the time zone and any time zone adjustments (such as Daylight Saving Time in the U.S.A.). After filling in the ``tm" structure, *localtime* sets the **tm_isdst**'th element of **tzname** to a pointer to an ASCII string that's the time zone abbreviation to be used with *localtime*'s return value.

Gmtime converts to Coordinated Universal Time.

Asctime converts a time value contained in a ``tm" structure to a 26−character string, as shown in the above example, and returns a pointer to the string.

Mktime converts the broken−down time, expressed as local time, in the structure pointed to by *tm* into a calendar time value with the same encoding as that of the values returned by the *time* function. The original values of the **tm_wday** and **tm_yday** components of the structure are ignored, and the original values of the other components are not restricted to their normal ranges. (A positive or zero value for **tm_isdst** causes *mktime* to presume initially that summer time (for example, Daylight Saving Time in the U.S.A.) respectively, is or is not in effect for the specified time. A negative value for **tm_isdst** causes the *mktime* function to attempt to divine whether summer time is in effect for the specified time.) On successful completion, the values of the **tm_wday** and **tm_yday** components of the structure are set appropriately, and the other components are set to represent the specified calendar time, but with their values forced to their normal ranges; the final value of **tm_mday** is not set until **tm_mon** and **tm_year** are determined. *Mktime* returns the specified calendar time; If the calendar time cannot be represented, it returns **−1**.

Difftime returns the difference between two calendar times, (*time1 − time0*), expressed in seconds.

Declarations of all the functions and externals, and the ``tm" structure, are in the **<time.h>** header file. The structure (of type) **struct tm** includes the following fields:

```
int tm_sec;      /* seconds (0 − 60) */
int tm_min;      /* minutes (0 − 59) */
int tm_hour;     /* hours (0 − 23) */
int tm_mday;     /* day of month (1 − 31) */
int tm_mon;      /* month of year (0 − 11) */
int tm_year;     /* year − 1900 */
int tm_wday;     /* day of week (Sunday = 0) */
int tm_yday;     /* day of year (0 − 365) */
int tm_isdst;    /* is summer time in effect? */
char *tm_zone;   /* abbreviation of timezone name */
long tm_gmtoff;  /* offset from UTC in seconds */
```

The *tm_zone* and *tm_gmtoff* fields exist, and are filled in, only if arrangements to do so were made when the library containing these functions was created. There is no guarantee that these fields will continue to exist in this form in future releases of this code.

Tm_isdst is non−zero if summer time is in effect.

Tm_gmtoff is the offset (in seconds) of the time represented from UTC, with positive values indicating east of the Prime Meridian.

SEE ALSO

getenv, strftime, time

NOTES

The return values point to static data; the data is overwritten by each call. The **tm_zone** field of a returned **struct tm** points to a static array of characters, which will also be overwritten at the next call.

clock − determine processor time used

SYNOPSIS

```
#include <time.h>

clock_t
clock(void)
```

DESCRIPTION

The **clock**() function determines the amount of processor time used since the invocation of the calling process, measured in CLOCKS_PER_SECs.

RETURN VALUES

The **clock**() function returns the amount of time used unless an error occurs, in which case the return value is −1.

STANDARDS

The **clock**() function conforms to ANSI X3.159−1989 (``ANSI C'').

strftime − format date and time

SYNOPSIS

```
#include <time.h>

size_t
strftime(char *buf, size_t maxsize, const char *format,
        const struct tm *timeptr)
```

DESCRIPTION

The **strftime**() function is not yet implemented.

The **strftime**() function formats the information from *timeptr* into the buffer *buf* according to the string pointed to by *format*.

The *format* string consists of zero or more conversion specifications and ordinary characters. All ordinary characters are copied directly into the buffer. A conversion specification consists of a percent sign `%' and one other character.

No more than *maxsize* characters will be placed into the array. If the total number of resulting characters, including the terminating null character, is not more than *maxsize*, **strftime**() returns the number of characters in the array, not counting the terminating null. Otherwise, zero is returned.

Each conversion specification is replaced by the characters as follows which are then copied into the buffer.

%A
is replaced by the locale's full weekday name.

%a
is replaced by the locale's abbreviated weekday name.

%B
is replaced by the locale's full month name.

%b
is replaced by the locale's abbreviated month name.

%c
is replaced by the locale's appropriate date and time representation.

%d
is replaced by the day of the month as a decimal number (01−31).

%H
is replaced by the hour (24−hour clock) as a decimal number (00−23).

%I
is replaced by the hour (12−hour clock) as a decimal number (01−12).

%j
is replaced by the day of the year as a decimal number (001−366).

%M
is replaced by the minute as a decimal number (00−59).

%m
is replaced by the month as a decimal number (01−12).

%p
is replaced by the locale's equivalent of either ``AM'' or ``PM''.

%S
is replaced by the second as a decimal number (00−60).

%U
is replaced by the week number of the year (Sunday as the first day of the week) as a decimal number (00−53).

%W
is replaced by the week number of the year (Monday as the first day of the week) as a decimal number (00−53).

%w
is replaced by the weekday (Sunday as the first day of the week) as a decimal number (0−6).

%X
is replaced by the locale's appropriate time representation.

%x
is replaced by the locale's appropriate date representation.

%Y
is replaced by the year with century as a decimal number.

%y
is replaced by the year without century as a decimal number (00−99).

%Z
is replaced by the time zone name.

%%
is replaced by `%'.

SEE ALSO

ctime, printf

STANDARDS

The **strftime**() function conforms to ANSI X3.159−1989 (``ANSI C'').

time − get time of day

SYNOPSIS

```
#include <time.h>

time_t
time(time_t *tloc)
```

DESCRIPTION

The **time**() function returns the value of time in seconds since 0 hours, 0 minutes, 0 seconds, January 1, 1970, Coordinated Universal Time.

A copy of the time value may be saved to the area indicated by the pointer *tloc*. If *tloc* is a NULL pointer, no value is stored.

Upon successful completion, **time**() returns the value of time. Otherwise a value of ((*time_t*) −1) is returned and the global variable *errno* is set to indicate the error.

ERRORS

The following error codes may be set in *errno*:

[EFAULT] An argument address referenced invalid memory.

SEE ALSO

ctime

HISTORY

A **time**() function appeared in Version 6 AT&T UNIX.

Introl Specific Functions

These are functions and header files that are used by the ANSI−C library. You may call the functions directly, but they are not part of ANSI−C. Some of the files are meant to be cusomized.

cvtflasc.c
: Convert a double to a string, this function is used indirectly by *printf*() through the *ofmt*() function described below.

crex.h
: The header file that can be included by a C program that the CREX realtime executive facilities directly.

doopen.c
: This function is called by the stdio functions to open a stream.

erase.c
: The is simple function sends a backspace, a space, and a backspace to erase a character on an output stream.

exitlist.c
: This file defines the structure that is used to record functions to be called when the program exits. *atexit*() is used to add functions to this list.

ifmt.c
: This function is used by the stdio functions, such as *scanf*() and *fscanf*(), that do formatted input.

introl.h
: This header file contains Introl specific definitions.

ioctl.c
: This functions is used to do stream specific control.

Makefile
: The Makefile for this library.

__main.c
: The *__main*() function is called by default at program startup to initialize the stdio functions. You can control whether __main() is called, and stdio initialized, at link time. You may not need any of the stdio functions in your target environment. See Configuring the runtime environment for more information.

ofmt.c
: This function is used by the stdio functions, such as *printf*() and *fprintf*(), that do formatted input.

tobase.c
: This function converts a integer to a digit in the given base.

68HC05 Support Functions

No C support library exists for the 68HC05.

The files referenced here are included only for documentation purposes. Some source files in this directory may not be included in this list but are retained for compatability previous releases of Introl−C.

In general it is unwise to call functions implemented here either from C or assembly language unless the functions have been defined in the ANSI−C, Introl−C, or Assembly library reference sections. The calling conventions, semantics, and even the existance of these functions may change from compiler release to compiler release.

68HC08 Support Functions

No C support library exists for the 68HC08.

The files referenced here are included only for documentation purposes. Some source files in this directory may not be included in this list but are retained for compatability previous releases of Introl−C.

In general it is unwise to call functions implemented here either from C or assembly language unless the functions have been defined in the ANSI−C, Introl−C, or Assembly library reference sections. The calling conventions, semantics, and even the existance of these functions may change from compiler release to compiler release.

6809 Support Functions

A pre−compiled C support library has been provided with Introl−CODE. The object code for the support library is supplied in the file $INTROL/lib/libc.a09.

The files referenced here are included only for documentation purposes. Some source files in this directory may not be included in this list but are retained for compatability previous releases of Introl−C.

In general it is unwise to call functions implemented here either from C or assembly language unless the functions have been defined in the ANSI−C, Introl−C, or Assembly library reference sections. The calling conventions, semantics, and even the existance of these functions may change from compiler release to compiler release.

fadd.s
: Floating point add.

fcmp.s
: Floating point compare.

fdeca.s
: Floating point decrement after.

fdiv.s
: Floating point divide.

fext.s
: Float to internal extended float conversion.

finca.s
: Floating point increment after.

fltint.s
: Convert float to integer.

fmul.s
: Floating point multiply.

fneg.s
: Floating point negate.

frexp.s
: The frexp() function.

ftst.s
: Floating point test and set flags.

intflt.s
: Convert integer to float.

io.c
: Sample input/output routines written in C.

jumps.s
: The setjmp() and longjmp() functions.

ladd.s
: Long integer addition.

land.s
: Long integer bitwise and.

lcmp.s
: Long integer comparison.

lcom.s
: Long integer one's complement.

ldeca.s

Long integer decrement after.

sldiv.s
: Long integer divide.

linca.s
: Long integer increment after.

lmul.s
: Long integer multiply.

lneg.s
: Long integer negation.

lor.s
: Long integer bitwise or.

lshl.s
: Long integer shift left.

lsub.s
: Long integer subtract.

ltst.s
: Long integer test and set flags.

lxor.s
: Long integer bitwise exclusive or.

Makefile
: The Makefile for this library.

sbrk.s
: The sbrk() function.

scdiv.s
: Signed 8 bit divide.

sdiv.s
: 16 bit signed and unsigned divide.

sexd.s
: Sign extend D into U:D.

slshr.s
: Signed long shift right.

smul.s
: 16 bit multiply.

ucdiv.s
: Unsigned 8 bit divide.

uldiv.s
: Unsigned long divide.

ulshr.s
: Unsigned long shift right.

68HC11 Support Functions

Several pre−compiled versions of the C support library have been provided with Introl−CODE. Their file names, descriptions, and the compiler command line required to recompile the components are listed in the table below. The object code for the support library is supplied in the files $INTROL/lib/libc.a11, libc.a01, libc.a03.

File name	Support library for:	Assembler command line
libc.a11	68HC11	cc11 *filename.s*
libc.a01	6801/03	cc11 −g01 *filename.s*
libc.a03	6301/03	cc11 −g03 *filename.s*

The files referenced here are included only for documentation purposes. Some source files in this directory may not be included in this list but are retained for compatability previous releases of Introl−C.

In general it is unwise to call functions implemented here either from C or assembly language unless the functions have been defined in the ANSI−C, Introl−C, or Assembly library reference sections. The calling conventions, semantics, and even the existance of these functions may change from compiler release to compiler release.

cdiv.s
: 8 bit divide.

clrup.s
: Clear the upper part of the long/float accumulator.

dinl.s
: Set the lower part of the long/float accumulator to D.

dinlt.s
: Set the upper part of the long/float accumulator to D.

idiv.s
: Signed and unsigned 16 bit divide.

dtol.s
: Load the long/float accumulator from *D.

dxtol.s
: Load the long/float accumulator from *(D+X).

extend.s
: Convert float to/from extended float format.

fabs.s
: Floating point absolute value.

fadd.s
: Floating point addition.

fcmp.s
: Floating point comparison.

fdeca.s
: Floating point decrement after.

fdecb.s
: Floating point increment before.

fdiv.s
: Floating point divide.

fftop.s
: Set top of long/float accumulator to 0xFFFF.

finca.s
: Floating point increment after.

fincb.s
: Floating point increment before.

fltint.s
: Convert float to integer.

fmul.s
: Floating point multiplication.

fneg.s
: Floating point negation.

frexp.s
: The frexp() function.

imul.s
: 16 bit multiply.

intflt.s
: Convert integer to float.

io.c
: Sample input/output routines written in C.

itol.s
: Convert integer to long.

jumps.s
: The setjmp() and longjmp() functions.

ladd.s
: Long integer addition.

land.s
: Long integer bitwise and.

lcmp.s
: Long integer comparison.

lcom.s
: Long integer one's complement.

ldeca.s
: Long integer decrement after.

ldecb.s
: Long integer decrement before.

sldiv.s
: Long integer divide.

lfacc.s
: Definition of the long/float accumulator (MC6801 and HD6301 only).

linca.s
: Long integer increment after.

lincb.s
: Long integer increment before.

lind.s
: Load the low part of the long/float accumulator into D.

lmul.s
: Long integer multiply.

lneg.s
: Long integer negate.

lor.s

Long integer bitwise or.

lshl.s
: Long integer shift left.

lshr.s
: Long integer shift right.

lsub.s
: Long integer subtract.

ltind.s
: Load the high part of the long/float accumulator into D.

ltod.s
: Store the long/float accumulator to *D.

ltodx.s
: Store the long/float accumulator to *(D+X).

ltos.s
: Push the long/float accumulator on the stack.

ltox.s
: Store the long/float accumulator to *X.

lxor.s
: Long integer bitwise exclusive or.

Makefile
: The Makefile for this library.

sbrk.s
: The sbrk() function.

sexd.s
: Sign extend D.

stol.s
: Pop the long/float accumulator from the stack.

tst.s
: Long integer test and set flags.

ulshr.s
: Unsigned long integer shift right.

xtol.s
: Load long/float accumulator from *X.

68HC12 Support Functions

A pre−compiled C support library has been provided with Introl−CODE. The object code for the support library is supplied in the file $INTROL/lib/libc.a12.

The files referenced here are included only for documentation purposes. Some source files in this directory may not be included in this list but are retained for compatability previous releases of Introl−C.

In general it is unwise to call functions implemented here either from C or assembly language unless the functions have been defined in the ANSI−C, Introl−C, or Assembly library reference sections. The calling conventions, semantics, and even the existance of these functions may change from compiler release to compiler release.

fabs.s
: Floating point absolute value.

fcmp.s
: Floating point compare.

fdeca.s
: Floating point decrement after.

finca.s
: Floating point increment after.

fladd.s
: Floating point addition.

fldiv.s
: Floating point division.

flext.s
: Convert float to/from internal extended.

flint.s
: Convert float to integer.

flmul.s
: Floating point multiply.

fneg.s
: Floating point negation.

frexp.s
: The frexp() function.

intfl.s
: Convert integer to float.

io.c
: Sample input/output function written in C.

jump.s
: The setjmp() and longjmp() functions.

ladd.s
: Long integer addition.

land.s
: Long integer bitwise and.

lcmp.s
: Long integer compare.

lcom.s
: Long integer one's complement.

ldeca.s

Long integer decrement after.

linca.s
Long integer increment after.

lmul.s
Long integer multiply.

lneg.s
Long integer negate.

lor.s
Long integer bitwise or.

lshl.s
Long integer shift left.

lshr.s
Long integer shift right.

lsub.s
Long integer subtract.

lxor.s
Long integer bitwise exclusive or.

Makefile
The Makefile for this library.

memset.s
The memset() function.

norm64.s
64 bit normalization routine.

retd4.s
Deallocate 4 bytes and return.

sbrk.s
The sbrk() function.

sldiv.s
Signed long integer divide.

strcat.s
The strcat() function.

strcmp.s
The strcmp() function.

strcpy.s
The strcpy() function.

strlen.s
The strlen() function.

switchb.s
8 bit switch table handler.

switchw.s
16 bit switch table handler.

uldiv.s
Unsigned long divide.

68HC16 Support Functions

A pre−compiled C support library has been provided with Introl−CODE. The object code for the support library is supplied in the file $INTROL/lib/libc.a16.

The files referenced here are included only for documentation purposes. Some source files in this directory may not be included in this list but are retained for compatability previous releases of Introl−C.

In general it is unwise to call functions implemented here either from C or assembly language unless the functions have been defined in the ANSI−C, Introl−C, or Assembly library reference sections. The calling conventions, semantics, and even the existance of these functions may change from compiler release to compiler release.

bankmac.mac
: Handy macros for K field manipulation.

dbabs.s
: Double absolute value.

dbadd.s
: Double addition.

dbcmp.s
: Double compare.

dbdadd.s
: Double update addition.

dbdeca.s
: Double decrement after.

dbdiv.s
: Double division.

dbext.s
: Convert double to/from internal extended.

dbfl.s
: Convert double to float.

dbfrexp.s
: The double frexp() function.

dbinca.s
: Double increment after.

dbint.s
: Convert double to integer.

dbldexp.s
: The double ldexp() function.

dbmul.s
: Double multiply.

dbneg.s
: Double negation.

dbpop.s
: Pop a double from the stack.

dbpush.s
: Puch a double on the stack.

dbstore.s
: Store double to *X.

dbtst.s

Test a double and set flags.

fladd.s
: Floating point addition.

flcmp.s
: Floating point comparison.

fldadd.s
: Floating point update addition.

fldb.s
: Convert float to double.

fldeca.s
: Floating point decrement after.

fldiv.s
: Floating point division.

fldsub.s
: Floating point update subtraction.

flext.s
: Convert float to/from internal extended.

flinca.s
: Floating point increment after.

flint.s
: Convert float to integer.

flmul.s
: Floating point multiply.

flneg.s
: Floating point negation.

fltst.s
: Test float and set flags.

intdb.s
: Convert integer to double.

intfl.s
: Convert integer to float.

io.c
: Sample input/output routines written in C.

lmul.s
: Long integer multiply.

Makefile
: The Makefile for this library.

memchr.s
: The memchr() function.

memcmp.s
: The memcmp() function.

memcpy.s
: The memcpy() function.

memcpy_d.s
: Register parameter version of memcpy().

memmove.s
: The memmove() function.

memset.s
: The memset() function.

norm32.s
: Normalize a 32 bit value.

norm64.s
: Normalize a 64 bit value.

norm128.s
: Normalize a 128 bit value.

pow2w.s
: 16 bit power of two table.

pow2b.s
: 8 bit power of two table.

retd2.s
: Return and deallocate two bytes.

retd4.s
: Return and deallocate four bytes.

sbrk.s
: The sbrk() function.

setjmp.s
: The setjmp() and longjmp() functions.

sldiv.s
: Signed long division.

slshr.s
: Signed long shift right.

strchr.s
: The strchr() function.

strcmp.s
: The strcmp() function.

strcpy.s
: The strcpy() function.

strlen.s
: The strlen() function.

strncat.s
: The strncat() function.

strncmp.s
: The strncmp() function.

strncpy.s
: The strncpy() function.

strrchr.s
: The strrchr() function.

switchb.s
: 8 bit switch table handler.

switchw.s
: 16 bit switch table handler.

uldiv.s
: Unsigned long division.

ulshr.s
: Unsigned long shift right.

xlmul.s
: 32 x 16 multiply.

xlshl.s
: Signed and unsigned long shift left.

68XXX Support Functions

Several pre−compiled versions of the C support library have been provided with Introl−CODE. Their file names, descriptions, and the compiler command line required to recompile the components are listed in the table below. The object code for the support library is supplied in the files $INTROL/lib/libc.a68, libc.a00, libc.a10, libc.a20, libc.a40, libcm.a20, and libcm.a40.

File name	Support library for:	Assembler command line
libc.a68	683XX	cc68 *filename.s*
libc.a00	68000	cc68 −g00 *filename.s*
libc.a10	68010	cc68 −g10 *filename.s*
libc.a20	68020/68030 with software floating−point	cc68 −g20 *filename.s*
libcm.a20	68020/68030 with 68881/2 floating−point	cc68 −g20 −gm *filename.s*
libc.a40	68040 with software floating−point	cc68 −g40 *filename.s*
libcm.a40	68040 with hardware floating−point	cc68 −g40 −gm5 *filename.s*

The files referenced here are included only for documentation purposes. Some source files in this directory may not be included in this list but are retained for compatability previous releases of Introl−C.

In general it is unwise to call functions implemented here either from C or assembly language unless the functions have been defined in the ANSI−C, Introl−C, or Assembly library reference sections. The calling conventions, semantics, and even the existance of these functions may change from compiler release to compiler release.

acos.s
: The acos() function.

asin.s
: The asin() function.

atan.s
: The atan() function.

cvtflasc.s
: The cvtflasc() function.

dbadd.s
: Double addition.

dbcmp.s
: Double comparison.

dbdiv.s
: Double division.

dbext.s
: Convert double to/from internal extended.

dbflt.s
: Convert double to float.

dbint.s
: Convert double to integer.

dbmul.s
: Double multiply.

exp.s
: The exp() function.

fabs.s
: The fabs() function.

fadd.s
: Floating point addition.

fcmp.s
: Floating point comparison.

fdiv.s
: Floating point division.

fext.s
: Convert float to/from internal extended.

floor.s
: The floor() function.

fltdb.s
: Convert float to double.

fltint.s
: Convert float to integer.

fmul.s
: Floating point multiply.

fneg.s
: Floating point negation.

frexp.s
: The frexp() function.

intdb.s
: Convert integer to double.

intflt.s
: Convert integer to float.

io.c
: Sample input/output functions written in C.

sldiv.s
: Long integer division.

lmul.s
: Long integer multiply.

log.s
: The log() function.

Makefile
: The Makefile for this library.

memcmp.s
: The memcmp() function.

memcpy.s
: The memcpy() function.

memset.s
: The memset() function.

modf.s
: The modf() function.

rts.s
: An rts instruction.

sbrk.s
: The sbrk() function.

setjmp.s

The setjmp() and longjmp() functions.

sin.s
: The sin() function.

sinh.s
: The sinh() function.

sqrt.s
: The sqrt() function.

strcat.s
: The strcat() function.

strcmp.s
: The strcmp() function.

strcpy.s
: The strcpy() function.

strncat.s
: The strncat() function.

strncmp.s
: The strncmp() function.

strncpy.s
: The strncpy() function.

tan.s
: The tan() function.

tanh.s
: The tanh() function.

uldiv.s
: Unsigned long integer divide.

Copyright