FC6809 INTROL-C

STANDARD LIBRARY
REFERENCE MANUAL

(FLEX)

The contents of this manual have been carefully reviewed and are believed to be entirely correct. However, Introl Corp. assumes no liability for inaccuracies.

The software described in this manual is proprietary and is furnished under a license agreement from Introl Corp. The software and supporting documentation may be used and/or copied only in accordance with said license agreement.

INTROL-C is a registered trademark of Introl Corp.
FLEX and UniFlex are trademarks of Technical Systems Consultants, Inc.
OS9 is a trademark of Microware Systems Corp.
UNIX is a trademark of Bell Laboratories

Introl Corp.
647 W. Virginia St.
Milwaukee, WI 53204 USA

tel. (414) 276-2937

FC6809 STANDARD LIBRARY

This manual describes each of the standard library routines supported by the FC6809 Introl-C Standard Library. The FC6809 Standard Library is usable with the Introl "fld' Loader for producing programs that are compatible with, and executable under, the Flex operating system. Note that Introl-C uses system call names which may differ from those used by your operating system. Those system calls which perform a function which is analogous to a recognized UNIX system call have been given the corresponding UNIX name rather than the name used by the particular operating system. The library functions appear in alphabetical order in this-manual.

> IMPORTANT NOTE: The majority of functions contained in the Standard Library have been pre-assigned a module "class number" of zero (0). Several "non-zero" class Standard Library modules are also included for user convenience, however, and are identified in the Appendix at the end of this Standard Library Manual. In general, these non-zero class modules are alternate forms of identically named class zero modules that exist in the library, modified to fit specific programming applications.

The following is a list of the functions included in this manual.

NAME

abs - integer absolute value

SYNOPSIS

```
int     abs(i)
int     i;
```

DESCRIPTION

abs returns the absolute value of its integer operand.

DIAGNOSTICS

SEE ALSO

NOTES

NAME

alloc - allocate memory

SYNOPSIS

```
char    *alloc(size)
int     size;
```

DESCRIPTION

alloc will attempt to allocate a block of memory whose size is given by the argument. If it is successful it returns a pointer to that memory otherwise it returns NULL.

DIAGNOSTICS

Returns NULL if the memory could not be allocated.

SEE ALSO

free(), sbrk()

NOTES

Alloc is an obsolete name for malloc(). It simply calls malloc() and returns.

NAME

atof - convert string to float

SYNOPSIS

```
float   atof(cptr)
char    *cptr;
```

DESCRIPTION

The atof function converts a string into a float which is then used as the return value of the function. The string should be null terminated although atof will stop reading the string as soon as an illegal character is reached. After ignoring preceding blanks the atof routine will convert as much of the string as conforms to normal floating point constant format to a floating point number. It will stop at the first character which is inconsistent with that format. If no floating point constant is found a 0 is returned.

A floating point constant consists of an integer part, a decimal point, a fractional part, and an exponential part. The integer and fractional parts may each consist of a string of one or more digits. The exponential part consists of an 'e' or 'E', followed by an optionally signed integer exponent. Either the integer or the fractional part (but not both) may be missing; either the decimal point or the exponential part (but not both) may be missing.

DIAGNOSTICS

SEE ALSO

atoi(), atol()

NOTES

Presently it is permitted to have spaces between the 'e' or 'E' and the first character of the integer representing the exponent.

NAME

atoi - convert string to integer

SYNOPSIS

```
int     atoi(ptr)
char    *ptr;
```

DESCRIPTION

Atoi's argument is a pointer to char which is assumed to point to a null terminated string which contains the ASCII representation of some integer number. The atoi function converts a string into an int which is the return value. The string should be null terminated although atoi will stop reading the string as soon as an illegal character is reached. After ignoring preceding blanks the atoi routine will convert as much of the string as conforms to normal integer constant format to an integer number. It will stop at the first character which is inconsistent with that format. If no integer constant is found a 0 is returned.

The integer constant format consists of an optional sign, followed by one or more digits. There should be no spaces interspersed within the number.

DIAGNOSTICS

SEE ALSO

atof(), atol()

NOTES

NAME

atol - convert string to long,

SYNOPSIS

```
long    atol(cptr)
char    *cptr;
```

DESCRIPTION

The atol function converts a string into a long which is the return value. The string should be null terminated although atol will stop reading the string as soon as an illegal character is reached. After ignoring preceding blanks the atol routine will convert as much of the string as conforms to normal long integer constant format to a long integer. It will stop at the first character which is inconsistent with that format. If no long integer constant is found a 0 is returned.

The long integer constant format consists of an optional sign, followed by one or more digits. There should be no spaces interspersed within the number.

DIAGNOSTICS

SEE ALSO

atof(), atoi()

NOTES

NAME

cprep - prepare environment for C program

SYNOPSIS

```
int     cprep(argc,argv,eext)
int     argc;
char    **argv;
char     *eext;
```

DESCRIPTION

Cprep first prepares the environment for the user C program and then call s "main", the usual entry-point to a user program. Cprep is usually referenced only from "cstart". The user program is not expected to make any explicit reference to this routine.

DIAGNOSTICS

SEE ALSO

cstart

NOTES

The result of an explicit reference to cprep is undefined.

NAME

cstart - runtime preparation routine

SYNOPSIS

DESCRIPTION

Cstart is a runtime preparation routine which is normally the first routine executed by an Introl-C program. Its only function is to set up the environment enough to allow the function "cprep" to be called. Cprep is a function which produces the runtime environment which is-expected by the user program. Cstart is included automatically by the linker. It is NOT expected that a user program will reference cstart explicitly via a function call.

DIAGNOSTICS

SEE ALSO

cprep()

NOTES

The result of an explicit reference to cstart is undefined.

NAME

ecvt - float to string conversion

SYNOPSIS

```
char    *ecvt(arg,ndigits,decpt,sign)
float   arg;
int     ndigits;
int     *decpt,*sign;
```

DESCRIPTION

This is a formatting routine used by printf for formatting floating point numbers in the e format.

Ecvt returns a pointer to a string which contains ascii characters representing a floating point number. The first argument is converted to a string whose length is indicated by the second argument. The third argument points to a variable in which the routine will write the location of the decimal point relative to the start of the string (negative numbers indicate that the decimal point is to the left of the first character of the string). The variable pointed to by the fourth argument is set nonzero if the float is negative otherwise it is set to zero.

The string is written in a static data area local to ecvt and is overwritten with the next call.

If the argument passed to ecvt is a legal floating point number the string will consist of a series of ascii digits terminated by a null. If the argument is out of the legal range for floats (as per the IEEE standard) the string will contain "NaN" (Not a Number). If the argument is either greater than the maximum or less than the minimum allowed for a float the characters "inf" (infinity) will be placed in the string (the fourth argument is set to indicate positive or negative infinity). The string itself contains neither a minus sign nor a decimal point nor a base ten exponent.

DIAGNOSTICS

SEE ALSO

fcvt(), itoa()

NOTES

NAME

execl - execute a program

SYNOPSIS

```
int     execl(cmd,arg0,arg1,...,0)
char    cmd,*arg0,*arg1,.....;
```

DESCRIPTION

Execl causes the present program to cease execution and a new program to execute. The name of the file to be executed must be contained in a string pointed to by the first argument. The additional arguments are assumed to be pointers to null terminated strings. These pointers will be passed to the program to be executed if they appeared as parameters on a command call line. The last argument MUST be a zero. The new process is given the arguments which follow the first argument in the execl call. The second argument of the execl call is the FIRST argument passed to the program to be executed (by convention referred to as argv(0). The last argument in the execl call must always be a zero.

DIAGNOSTICS

This function NEVER returns.

SEE ALSO

NOTES

The sum total of lengths of the argument strings (including a space to be placed between each argument) must not exceed the length of a FLEX line buffer, which is 128 bytes long.

NAME

exit - exit a program with file cleanup

SYNOPSIS

```
int     exit(stat)
int     stat;
```

DESCRIPTION

Exit aborts a C program and returns to the operating system. The status value is returned to the operating system. Exit also flushes any open file buffers and closes all open files before exiting.

DIAGNOSTICS

SEE ALSO

_exit()

NOTES

NAME

_exit - exit a program without file cleanup

SYNOPSIS

```
int     _exit(stat)
int     stat;
```

DESCRIPTION

_exit aborts a C program and returns to the operating system. The status value is returned to the operating system. The _exit routine does not explicitly flush the file buffers.

DIAGNOSTICS

SEE ALSO

exit()

NOTES

NAME

_extend - extend float

SYNOPSIS

```
int     _extend(f,ef)
float   f;
struct  extflt
        {
        char    sign;
        int     exp;
        long    mantissa
        }  *ef;
```

DESCRIPTION

_extend extends a floating point number (its first argument) and stores the result in the structure pointed to by the second argument. The first element of the structure contains the sign bit of the number, the second element contains the unbiased exponent, and the thirs element contains the mantissa.

DIAGNOSTICS

SEE ALSO

_unext()

NOTES

NAME

fclose - close file

SYNOPSIS

```
#include  "stdio.h"
int    fclose(fp)
FILE   *fp;
```

DESCRIPTION

Fclose will close the file indicated by its argument. The argument must be a file pointer which was previously returned from an fopen unless it is STDIN, STDOUT, or STDERR. If the file has been opened for writing, fclose will automatically flush the remaining contents of the buffer.

DIAGNOSTICS

fclose will return ERROR if the file could not be closed. The external variable "errno" will contain the error code which was returned by the operating system..

SEE ALSO

fgets(), fopen(), fprintf(), fputs(), fscanf(), getc()

NOTES

NAME

fcvt - float to string conversion

SYNOPSIS

```
char   *fcvt(arg,ndigits,decpt,sign)
float  arg;
int    ndigits;
int    *decpt,*sign;
```

DESCRIPTION

This is a formatting routine used by printf for formatting floating point numbers in the f format. It is similar to the "ecvt" routine except that the correct digit will be rounded as demanded by Fortran F-format for the number of digits indicated by the second argument

Fcvt returns a pointer to a string which contains ascii characters representing a floating point number. The first argument is converted to a string whose length is indicated by the second argument. The third argument points to a variable in which the routine will write the location of the decimal point relative to the start of the string (negative numbers indicate that the decimal point is to the left of the first character of the string). The variable pointed to by the fourth argument is set nonzero if the float is negative; otherwise it is set to zero.

The string is written in a static data area local to fcvt and is overwritten with the next call.

If the argument passed to fcvt is a legal floating point number the string will consist of a series of ascii digits terminated by a null. If the argument is out of the legal range for floats (as per the IEEE standard) the string will contain "NaN' (Not a Number). If the argument is either greater than the maximum or less than the minimum allowed for a float the characters "inf" (infinity) will be placed in the string (the fourth argument is set to indicate positive or negative infinity). The string itself contains neither a minus sign nor a decimal point nor a base ten exponent.

DIAGNOSTICS

SEE ALSO

ecvt(), itoa()

NOTES

NAME

fgets - read file into string

SYNOPSIS

```
#include "stdio.h"
int     fgets (s,n,fp)
char   *S;
int    n;
FILE   *fp;
```

DESCRIPTION

Fgets will read a line of up to n characters from the file pointed to by its third argument into the area pointed to by its first argument. Its third argument must be a file pointer previously returned by an fopen call. Fgets returns a pointer to the start of the line read or NULL if for some reason no line could be read. The function reads the number of characters indicated by its second argument or until an end of line is encountered, whichever comes first. The trailing newline IS included in the line read.

DIAGNOSTICS

fgets will return NULL if the file could not be read from; this is usually interpreted as an End Of File.

SEE ALSO

fclose(), fflush(), fopen(), fprintf(), fputs(), fscanf(), getc(), gets()

NOTES

If there is a trailing newline character read from the file fgets will include it in the string whereas gets will not.

NAME

_filespec - Build file specification

SYNOPSIS

```
*include "stdio.h"
int     _filespec(n,fp,ext)
char    *n;
FILE    *fp;
char    ext;
```

DESCRIPTION

The _filespec function builds a file specification in the fcp pointed to by the second argument. The first argument points to a file name string that may contain a drive specifier and an extension. If no drive is given in the name, the system working disk is assumed. If no extension is given in the name, the value of the third argument is used in a call to the FLEX routine SETEXT to set the default extension. (see "The FLEX Advanced Programmers Guide" for more details on the ext parameter.)

DIAGNOSTICS

Returns ERROR if a valid file specification could not be made.

SEE ALSO

NOTES

This routine is used internally by some of the file routines and is not guaranteed to be supported in the future.

NAME

_fms - Call to FLEX FMS entry point

SYNOPSIS

```
#include "stdio.h"
int     _fms(fp,c)
FILE    *fp;
char    C;
```

DESCRIPTION

This is a short assembly language routine that allows a C program to call the FLEX FMS entry point. The desired function should be placed in fp->f.function (see the flex.h header file). The value of the second parameter is placed in the A accumulator before the call to the FMS entry point. On return, fms returns an integer representing the value of the A Accumulator or ERROR.

DIAGNOSTICS

Returns ERROR if FLEX detected an error in the FMS call.

SEE ALSO

NOTES

This routine is used internally by some of the file routines and is not guaranteed to be supported in the future.

NAME

fopen - open a file

SYNOPSIS

```
#include        "stdio.h"
FILE    *fopen(name,mode)
char    *name,*mode
```

DESCRIPTION

Fopen will open the file whose name is pointed to by its first argument with the attributes indicated in the string pointed to by its second argument. It returns a value of type pointer to FILE which must be used as an argument on subsequent references to the file.

The options with which the file is to be opened are specified as ASCII characters in the mode string (whose pointer is passed as the second parameter). One of the characters in this string indicates the mode for which the file will be opened. The appropriate modes are:

r - read: File is opened for read access

w - write: File is opened for write access

If neither of these characters appears in the string the file is opened for read access. The result of placing more than one of these characters in the string is undefined.

In addition to one of the preceding characters a b may appear in the string. The 'b' option indicates that the file is a binary file while the absence of a 'b' indicates that the file should be opened as a text file.

DIAGNOSTICS

Fopen will return ERROR if the file could not be opened and the external variable "errno" will contain any error code returned by the system.

SEE ALSO

fclose(), fgets(), fprintf(), fputs(), fscanf(), getc()

NOTES

The current version of fopen returns ERROR when it fails to open a file rather than the more common return value of NULL.

NAME

fprintf - formatted output conversion

SYNOPSIS

```
#include      "stdio.h"
int      fprintf(stream,control [,arg])
FILE   *stream;
char   *control;
```

DESCRIPTION

Fprintf is nearly identical to printf except that here the output file specification is explicitly given as the first argument. All output is sent to the file pointed to by the first argument. The parameters to fprintf consist of pointer to FILE, followed by a pointer to a null terminated string, followed by zero or more arguments. fprintf formats and writes the arguments following the control string using the control string to direct formatting and conversion. The control string may contain normal characters (which are simply copied to the output file) and conversion specifications which control the writing of the arguments. Each conversion provides information used to format its corresponding argument following the control string. Conversion specifications begin with a percent character (%), perhaps followed by some options and terminated by a conversion character. All the options are, of course, optional but those that are included must appear in the specified order. The legal options (in the order they must appear) are as follows:

Dash (-): indicates that if the number to be written is shorter than the specified field length that it should be left justified. If this option is omitted the number will be right justified.

Zero (0): indicates that if the number to be written is shorter than the specified field length that it should be padded with zeros to fill the field length. If this option is omitted the field will be padded with blanks.

Digit string: indicates the minimum field width. The argument will be written in a field at least this wide. This field may be replaced with a star (*) which will cause the field width to be taken from the next corresponding argument (of type integer) in the argument list.

Period (.): separates the field width from the next digit string.

Digit string: indicates the precision. For a float the precision is the number of digits to be written to the right of the decimal point. For a string the precision is the maximum number of characters which will be written. This field may be replaced with a star (*) which will cause the field width to be taken from the next

corresponding argument (assumed to be an integer) in the argument list

Long (l): (letter ell) indicates that the corresponding argument is to be written as a long rather than an int.

The valid conversion characters and their meanings are as follows:

d The argument is assumed to be of type int and is written in decimal notation.

o The argument is written in octal (without leading 0).

x Argument is written in hexadecimal (without leading Ox).

u The argument is assumed to be unsigned and written in decimal notation.

c The argument is written as a character.

s The argument is assumed to be a pointer to a null terminated string. Characters are copied from the control string to the output string until a null character is reached or until the number of characters given by the precision are copied. The terminating null is not copied.

e The argument is assumed to be a float and written out in a decimal notation of the following form: [-d.dddddde[+|-]dd That is a negative sign if the number is negative, a single digit, followed by a decimal point, followed by several digits, followed by an 'e', followed by a sign, followed by two digits.

f The argument is assumed to be a float and written out in a decimal notation of the following form: [-]ddd.dddd where the length of the string of digits following the decimal point is given by the precision.

g Prints in either e or f format; whichever is shorter.

If a character which is neither an option nor a conversion character is found while scanning a conversion specification the character following the percent sign (%) is simoly written and no conversion specification is assumed. Thus to write a percent sign one writes it twice(%%).

DIAGNOSTICS

Fprintf returns ERROR if it fails.

SEE ALSO

printf(),sprintf()

NAME

fputs - write a string to a file

SYNOPSIS

```
#include  "stdio.h"
int      fputs(s,fp)
char     *S;
FILE     *fp;
```

DESCRIPTION

Fputs copies the string pointed to by the first argument to the file indicated by the second argument. The second argument of type pointer to FILE and should have been returned by a call to fopen unless it is STDOUT or STDERR.

DIAGNOSTICS

Returns ERROR if an error occurred while attempting to write the string.

SEE also

puts()

NOTES

NAME

free - free memory

SYNOPSIS

```
char    *free(block)
char    *block;
```

DESCRIPTION

Free will attempt to free a block of memory indicated by its argument. The only valid argument for free is a pointer previously returned by an alloc call. This routine should only be used to free a block that has been allocated via alloc. The result of freeing the same block of memory more than once or attempting to use, as an argument, a pointer which was not returned by an alloc call is undefined (bad things happen).

DIAGNOSTICS

SEE ALSO

alloc(), sbrk()

NOTES

NAME

fscanf - formatted input conversion

SYNOPSIS

```
#include      "stdio.h"
int     fscanf(file,control [,pointer1]...)
FILE    *file;
char    *control;
```

DESCRIPTION

Fscanf is nearly identical to scanf except that the input file specification is explicitly stated; the input is taken from the file pointed to by the first argument. The parameters to fscanf consist of a pointer to file, followed by a pointer to a null terminated string (the control string), followed by zero or more arguments of type pointer. Fscanf reads groups of characters from the input file pointed to by the first argument, interprets them according to the control string, and writes the results into the arguments pointed to by their corresponding argument pointers. The control string may contain blanks, tabs, and newlines which match optional white space in the input; it may contain ordinary characters which must match the input string exactly character per character; and it may contain conversion specifications used to control the interpretation of the input stream. Each conversion specification provides information used to translate a segment of the input stream into a value which may then be placed into an argument pointed to by its corresponding pointer in the argument list.

Conversion specifications begin with a percent character perhaps followed by some options, and terminated by a conversion character. All the options are, of course, optional but those that are included must appear in the specified order. The legal options (in the order they must appear) are:

Star (*): indicates that this conversion specification has no corresponding pointer in the argument list. This effectively skips a value in the input stream.

Digit string: indicates the maximum field width; the maximum number of characters which this conversion specification will cause to be read from the input stream.

Long (l): (letter ell) indicates that the corresponding pointer is pointing to a long rather than an int. This has no effect when preceding an e or f.

The valid conversion characters and their meanings are as follows:

d A decimal integer is expected in the input string. Its corresponding pointer is assumed to be of type *int.

o An octal integer is expected in the input string. Its corresponding pointer is assumed to be of type *int.

x A hexadecimal integer is expected in the input string. Its corresponding pointer is assumed to be of type lint.

h A decimal integer is expected in the input string. Its corresponding pointer is assumed to be of type short.

u An unsigned decimal integer is expected in the input string. Its corresponding pointer is assumed to be of type *unsigned.

c The very next character is read from the input string (even if it's a blank). Its corresponding pointer is assumed to be of type char.

s A string is expected in the input string. Its corresponding pointer is assumed to be of type *char. It should point to a space large enough to hold the input string plus an added null. Characters are read, starting with the next nonblank character, until the number of characters given in the precision is reached or until a blank, tab, or newline is reached.

e (same as f)

f A floating point number is expected in the input string. Its corresponding pointer is assumed to be of type *float.

DIAGNOSTICS

The return value of this function is the number of parameters that were matched (read in from the input line) or EOF (-1).

SEE ALSO

scanf(), sscanf()

NOTES

Exactly one line of input is consumed for each call to fscanf. Thus fscanf will not fetch a new line even though there are still conversion specifications left to process nor will it save any input left from the preceding line for the next call to fscanf.

A hexadecimal number may not be preceded by a 0x.

Any character within a conversion specifier which is not a legal conversion specifier option or conversion character will be ignored along with the preceding percent sign and any characters inbetween. Thus there is no way to match a '%' on the input line.

NAME

getc - get the next character from a file

SYNOPSIS

```
#include "stdio.h"
int     getc(fp)
FILE    fp;
```

DESCRIPTION

Getc returns the next character from the file indicated by its argument. Its argument is of type pointer to FILE and should have been previously returned from an fopen call unless it is STDIN.

DIAGNOSTICS

Getc returns ECF (-1) upon reading end of file or on error.

SEE ALSO

getchar()

NOTES

Notice the return value of getc is an integer not a character. This is so that getc can return ECF (-1) on end of file.

NAME

getchar - get a character from the standard input

SYNOPSIS

```
int     getchar()
```

DESCRIPTION

Getchar is identical to getc(stdin). It returns the next character from the standard input.

DIAGNOSTICS

Getchar returns ECF (-1) upon reading end of file or on error.

SEE ALSO

getc()

NOTES

Notice the return value of getchar is an integer not a character. This is so that getchar can return an ECF (-1) on end of file.

NAME

_getchr - Call FLEX GETCHR entry point.

SYNOPSIS

```
#include "stdio.h"
int      _getchr()
```

DESCRIPTION

This function returns the value obtained by a call to the FLEX entry point GETCHR (get console character).

DIAGNOSTICS

SEE ALSO

NOTES

This routine is used internally by some of the file routines and is not guaranteed to be supported in the future.

NAME

gets - read input into string

SYNOPSIS

```
int        gets(s)
char       *S;
```

DESCRIPTION

Gets will read a line from the standard input (STDIN) into the area pointed to by its argument. Gets returns a pointer to the start of the line read, or NULL if for some reason no line could be read. The function reads until an end of line is encountered. The trailing newline is NOT included in the line read (compare this with fgets(s,n,stdin)).

DIAGNOSTICS

Gets will return NULL on end of file and error.,

SEE ALSO

Fclose(), fflush(), fgets(), fopen(), fprintf(), fputs(), fscanf(), getc().

NOTES

Gets will not include any trailing newline character in the string whereas fgets will.

NAME

index - find first occurrence of character

SYNOPSIS

```
int     index(s,c)
char    *s;
char c;
```

DESCRIPTION

Index searches the string whose pointer is passed as its first argument and returns a pointer to the first occurrence of the character specified by the second argument. A zero is returned if the character does not appear in the string.

DIAGNOSTICS

SEE ALSO

rindex()

NOTES

NAME

isalpha - test for alpha character

SYNOPSIS

```
int     isalpha(ch)
char    ch;
```

DESCRIPTION

Returns true (non zero) if its argument is an alpha character (a through z or A through Z); otherwise returns false (zero).

DIAGNOSTICS

SEE ALSO

isdigit(), islower(), isspace(), isupper()

NOTES

NAME

isdigit - test for digit

SYNOPSIS

```
int     isdigit(ch)
char    ch;
```

DESCRIPTION

Returns true (non zero) if its argument is a digit (0 through 9); otherwise returns false (zero).

DIAGNOSTICS

SEE ALSO

isalpha(), islower(), isspace(), isupper()

NOTES

NAME

islower - test for lower case

SYNOPSIS

```
int     islower(ch)
char    ch;
```

DESCRIPTION

Returns true (non zero) if its argument is a lower case alpha character (a through z); otherwise returns false (zero).

DIAGNOSTICS

SEE ALSO

isalpha(), isdigit(), isspace(), isupper()

NOTES

NAME

isspace - test for white space

SYNOPSIS

```
int     isspace(ch)
char    ch;
```

DESCRIPTION

Returns true (non zero) if its argument is a space, tab, or newline character; otherwise returns false (zero).

DIAGNOSTICS

SEE ALSO

isalpha(), isdigit(), islower(), isupper()

NOTES

NAME

isupper - test for upper case

SYNOPSIS

```
int     isupper(ch)
char    ch;
```

DESCRIPTION

Returns true (non zero) if its argument is an upper case alpha character (A through Z); otherwise returns false (zero).

DIAGNOSTICS

SEE ALSO

isalpha(), isdigit(), islower(), isspace()

NOTES

NAME

itoa - convert integer to ascii string

SYNOPSIS

```
int     itoa(n,s)
int     n;
char    *S;
```

DESCRIPTION

Itoa converts its first argument into a null terminated ascii string which is stored at the location pointed to by its second argument. If the integer is negative the string will be preceded by a minus sign. The second argument should point to an area large enough to contain the resultant string which may contain a sign, up to 5 digits, and a NULL termination character.

DIAGNOSTICS

SEE ALSO

fcvt(), ecvt()

NOTES

NAME

longjmp - non-local goto

SYNOPSIS

```
#include     "stdio.h"
int     longjmp(envp,n)
struct jmp_buf *envp;
int     n;
```

DESCRIPTION

Longjmp works in conjunction with setjmp to provide the ability to jump outside of a function. Compare this to a normal goto for which the destination must be in the same function as the goto statement. Setjmp is used to mark a location as a destination (that is save a copy of the current environment) for later use by the longjmp routine. The argument to setjmp is a pointer to structure which will hold the current environment. A pointer to this structure is used as an argument to longjmp. Longjmp simply restores the environment which was saved by the setjmp call. The effect is that execution continues at the location where the environment was saved (inside the setjmp call). The appearance is that of a return from setjmp.

To mark a location one makes a call to setjmp. This will initialize the contents of the structure whose pointer was passed as an argument. From this call, setjmp will return the value 0. Later, when control is returned here from a longjmp, the return value will be decided by the second argument of the longjmp call.

Now a jump can be made to this location by making a call to longjmp, using a pointer to the same structure that was initialized by setjmp as the first argument and an integer as the second argument. The second argument, will be used as the return value when control is transferred to the setjmp environment

The destination of a longjump must be in a function which has not itself returned inbetween the call to setjmp and the call to longjmp. That is, the destination of a longjmp must be within a currently active function.

DIAGNOSTICS

SEE ALSO

NOTES

NAME

malloc allocate memory

SYNOPSIS

```
char    *malloc(size)
int     size;
```

DESCRIPTION

malloc will attempt to allocate a block of memory whose size is given by the argument. If it is successful it returns a pointer to that memory, otherwise it returns NULL.

DIAGNOSTICS

Returns NULL if the memory could not be allocated.

SEE ALSO

free(), sbrk()

NOTES

NAME

max - return the maximum of two values

SYNOPSIS

```
int     max(a,b)
int     a,b;
```

DESCRIPTION

Max returns the greater of its two arguments.

DIAGNOSTICS

SEE ALSO

min()

NOTES

NAME

min - return the minimum of two values

SYNOPSIS

```
int     min(a,b)
int     a,b;
```

DESCRIPTION

Min returns the lesser of its two arguments.

DIAGNOSTICS

SEE ALSO

max()

NOTES

NAME

modf - return fractional part of float

SYNOPSIS

```
float   modf(fp,fint)
float   fp;
float   *fint;
```

DESCRIPTION

Modf takes a floating point number as its first argument and returns its fractional part. Its nonfractional part is written to the location pointed to by the second argument.

This routine is used by ecvt and fcvt.

DIAGNCSTICS

SEE ALSO

NOTES

NAME

movmem - copy a block of memory from one location to another

SYNOPSIS

```
int     movmem (from,to,length)
char    *from, *to;
unsigned        length;
```

DESCRIPTION

Movmem copies the number of bytes given by the third argument from the location pointed to by first argument to the location pointed to by the second argument. The new copy will exactly reflect the original as it existed before the call even if the two blocks of memory overlap (in that case, of course, the original will be partially overwritten).

DIAGNOSTICS

SEE ALSO

NOTES

NAME

printf - formatted output conversion

SYNOPSIS

```
int     printf(control [,arg]...)
char    *control;
```

DESCRIPTION

Printf is nearly identical to fprintf excect that there is no output file specification explicitly stated; the result is written to stdout. The parameters to printf consist of a pointer to a null terminated string followed by zero or more arguments. Printf formats and writes the arguments following the control string using the control string to direct formatting and conversion. The control string may contain normal characters (which are simply copied to the output file) and conversion specifications which control the writing of the arguments. Each conversion specification provides information used to format its corresponding argument following the control string. Conversion specifications begin with a percent character (%), perhaps followed by some options and terminated by a conversion character. All the options are, of course, optional but those that are included must appear in the specified order. The legal options (in the order they must appear) are as follows:

Dash (-): indicates that if the number to be written is shorter than the specified field length, it should be left justified. if this option is omitted the number will be right justified.

Zero (0): indicates that if the number to be written is shorter than the specified field length, it should be padded with zeros to fill the field length. If this option is omitted the field will be padded with blanks.

Digit string: indicates the minimum field width. The argument will be written in a field at least this wide. This field may be replaced with a star (*) which will cause the field width to be taken from the next corresponding argument (assumed to be an integer) in the argument list.

Period (.): separates the field width from the next digit string.

Digit string: indicates the precision. For a float the precision is the number of digits to be written to the right of the decimal point. For a string the precision is the maximum number of characters which will be written. This field may be replaced with a star (*) which will cause the field width to be taken from the next corresponding argument (assumed to be an integer) in the argument list.

Long (l): (letter ell) indicates that the corresponding argument is to be written as a long rather than an int.

The valid conversion characters and their meanings are as follows:

d The argument is assumed to be of type int and is written in decimal notation.

o The argument is written in octal (without leading 0).

x Argument is written in hexadecimal (without leading Ox).

u The argument is assumed to be unsigned and written in decimal notation.

c The argument is written as a character.

s The argument is assumed to be a pointer to a null terminated string. Characters are copied from the control string to the output string until a null character is reached or until the number of characters given by the precision are copied. The terminating null is not copied.

e The argument is assumed to be a float and written out in a decimal notation of the following form: [-]d.dddddde[+|-]dd That is a negative sign if the number is negative, a single digit, followed by a decimal point, followed by several digits, followed by an 'e', followed by a sign, followed by two digits.

f The argument is assumed to be a float and written out in a decimal notation of the following form: [-]ddd.dddd where the length of the string of digits following the decimal point is given by the precision.

g Prints in either e or f format; whichever is shorter.

If a character which is neither an option nor a conversion character is found while scanning a conversion specification the character following the percent sign (%) is simply written and no conversion specification is assumed. Thus to print out a percent sign one writes it twice (%%). A space is NOT a legal option.

DIAGNOSTICS

Printf returns ERROR if it fails.

SEE ALSO

fprintf(), sprintf()

NOTES

NAME

putc - write a character to a file

SYNOPSIS

```
#include  "stdio.h"
int       putc(c,fp)
char      c;
FILE      *fp;
```

DESCRIPTION

Putc sends the character given as its first argument to the file whose file pointer is given as its second argument. The file pointer must have been previously returned from an fopen call unless it is STDOUT or STDERR.

DIAGNOSTICS

Putc returns ERROR (-1) if an error occurs during the write process.

SEE ALSO

NOTES

NAME

putchar - write a character to the standard output

SYNOPSIS

```
int     putchar(c)
char    C;
```

DESCRIPTION

Putchar sends the character given as its argument to STDOUT. A call of the form putchar(c) is identical to putc(c,stdout).

DIAGNOSTICS

Putchar returns ERROR (-1) if an error occurs during the write process.

SEE ALSO

putc()

NOTES

NAME

putchr - Call FLEX PUTCHR entry point.

SYNOPSIS

```
#include "Istdio.h"
int      _putchr(c)
char     c;
```

DESCRIPTION

This function performs a call to the FLEX entry point PUTCHR to perform console output.

DIAGNOSTICS

SEE ALSO

NOTES

This routine is used internally by some of the file routines and is not guaranteed to be supported in the future.

NAME

puterr - write a char to the standard error output (STDERR)

SYNOPSIS

```
int     puterr(c)
char    c;
```

DESCRIPTION

Puterr sends the character given as its argument to STDERR. A call of the form puterr(c) is identical to putc(c,stderr).

DIAGNOSTICS

Puterr returns ERROR (-1) if an error occurs during the write process.

SEE ALSO

NOTES

STDERR is always directed to the terminal.

NAME

puts - write a string to standard output

SYNOPSIS

```
int    puts(s)
char   *s;
```

DESCRIPTION

Puts copies the string pointed to by the argument to the standard output. The effect is the same as fputs(s,stdout).

DIAGNOSTICS

Returns ERROR if an error occurred while attempting to write the string.

SEE ALSO

fputs()

NOTES

Does NOT append a newline (contrary to some implementations).

NAME

reverse - reverse a string in place

SYNOPSIS

```
int     reverse(s)
char    *s;
```

DESCRIPTION

Reverses the order of the elements of a string pointed to by the argument. If the string the argument pointed to was "abcdef" before the call, it would be "fedcba" after the call.

DIAGNOSTICS

SEE ALSO

NOTES

NAME

rewind - reset specified file to beginning

SYNOPSIS

```
#include "stdio.h"
int      rewind(fp)
FILE     *fp;
```

DESCRIPTION

Rewind resets the file back to the beginning.

DIAGNOSTICS

Returns ERROR for improper file specification.

SEE ALSO

NOTES

NAME

rindex - find last occurrence of character

SYNOPSIS

```
int     rindex(s,c)
char    *S;
char c;
```

DESCRIPTION

Rindex searches the string whose pointer is passed as its first argument and returns a pointer to the last occurrence of the character specified by the second argument. A zero is returned if the character does not appear in the string.

DIAGNOSTICS

SEE ALSO

index()

NOTES

NAME

sbrk - allocate memory

SYNOPSIS

```
char     *sbrk(size)
int      size;
```

DESCRIPTION

Sbrk will attempt to allocate a block of memory whose size is given by the argument. If it is successful it returns a pointer to that memory; otherwise it returns ERROR.

Sbrk is similar to alloc except that there is no way to return the memory to the system.

DIAGNOSTICS

Returns ERROR (-1) if the memory could not be allocated.

SEE ALSO

alloc(), brk(), free()

NOTES

NAME

scanf - formatted input conversion

SYNOPSIS

```
int    scanf(control [,pointer1] ... )
char   *control;
```

DESCRIPTION

Scanf is nearly identical to fscanf except that there is no input file specification explicitly stated; the input is taken from stdin. The parameters to scanf consist of a pointer to a null terminated string (the control string) followed by zero or more arguments of type pointer. Scanf reads groups of characters from the standard input, interprets them according to the control string and writes the results into the arguments pointed to by their corresponding argument pointers. The control string may contain blanks, tabs, and newlines which match optional white space in the input; it may contain ordinary characters which must match the input string exactly character per character; and it may contain conversion specifications used to control the interpretation of the input stream. Each conversion specification provides information used to translate a segment of the input stream into a value which may then be placed into an argument pointed to by its corresponding pointer in the argument list. Conversion specifications begin with a percent character (%), perhaps followed by some options, and terminated by a conversion character. All the options are, of course, optional but those that are included must appear in the specified order.

The legal options (in the order they must appear) are as follows:

Star (*): indicates that this conversion specification has no corresponding pointer in the argument list. This effectively skips a value in the input stream.

Digit string: indicates the maximum field width; the maximum number of characters which this conversion specification will cause to be read off the input stream.

Long (letter ell) indicates that the corresponding pointer is pointing to a long rather than an int. This has no effect when preceding an e or f.

The valid conversion characters and their meanings are as follows:

d A decimal integer is expected in the input string. Its corresponding pointer is assumed to be of type *int.

o An octal integer is expected in the input string. Its corresponding pointer is assumed to be of type *int.

x A hexadecimal integer is expected in the inout string. Its corresponding pointer is assumed to be of type lint.

h A decimal integer is expected in the input string. Its corresponding pointer is assumed to be of type short.

u An unsigned integer is expected in the input string. Its corresponding pointer is assumed to be of type *unsigned.

c The very next character is read from the input string (even if it's a blank). Its corresponding pointer is assumed to be of type *char.

s A string is expected in the input string. Its corresponding pointer is assumed to be of type *char. It should point to a space large enough to hold the input string plus an added null. Characters are read, starting with the next nonblank character, until the number of characters given in the precision is reached or until a blank, tab, or newline is reached.

e (same as f)

f A floating point number is expected in the input string Its corresponding pointer is assumed to be of type *float.

The return value of this function is the number of parameters that were matched (read in off the input line) or ECF.

DIAGNOSTICS

SEE ALSO

fscanf(), sscanf()

NOTES

Exactly one line of input is consumed for each call to scanf. Thus scanf will not fetch a new line even though there are still conversion specifications left to process nor will it save any input left from the preceding line for the next call to scanf. If, for example, one makes a call to scanf with a control string which indicates 3 arguments are expected while only 2 appear on the input line scanf will NOT continue to read lines. Fscanf will simply return with a value of 2. Likewise if the input line had contained 4 arguments only 3 would have been read while the fourth would be discarded.

A hexadecimal number may not be preceded by a Ox.

Any character within a conversion specifier which is not a legal conversion specifier option or conversion character will be ignored along with the preceding percent sign and any characters in between. Thus there is no way to match a '%' on the input line.

NAME

_setext - Call FLEX SETEXT entry point

SYNOPSIS

```
#include "stdio.h"
int     _setext(fp,ext)
FILE     fp;
char    ext;
```

DESCRIPTION

The _setext function performs a call to the FLEX routine SETEXT to set a default file name extension into the given file control block.

DIAGNOSTICS

SEE ALSO

NOTES

This routine is used internally by some of the file routines and is not guaranteed to be supported in the future.

NAME

setjmp - non-local goto

SYNOPSIS

```
#include
int     setjmp (envp)
       jmp_buf *envp;
```

DESCRIPTION

Setjmp works in conjunction with longjmp to provide the ability to jump outside of a function. Compare this to a normal goto for which the destination must be in the same function as the goto statement. Setjmp is used to mark a location as a destination (that is save a copy of the current environment) for later use by the longjmp routine. The argument to setjmp is a pointer to structure which will hold the current environment. A pointer to this structure is used as one of the arguments to longjmp. Longjmp simply restores the environment which was saved by the setjmp call. The effect is that execution continues at the location where the environment was saved (inside the setjmp call). The appearance is that of a return from setjmp.

To mark a location one makes a call to setjmp. This will initialize the contents of the structure whose pointer was passed as an argument. From this call setjmp will return the value 0. Later, when control is returned here from a longjmp, the return value will be decided by the second argument of the longjmp call. (see longjmp)

Now a jump can be made to this location by making a call to longjmp using a pointer to the same structure that was initialized by setjmp as the first argument and an integer as the second argument. The second argument will be used as the return value when control is transferred to the setjmp environment.

The destination of a longjmp must be in a function which has not itself returned inbetween the call to setjmp and the call

to longjmp.

DIAGNOSTICS

SEE ALSO

longjmp()

NOTES

NAME

sprintf - formatted output conversion

SYNOPSIS

```
int     sprintf(string,control [,arg1]...)
char    *string, *control;
```

DESCRIPTION

Sprintf is nearly identical to printf except that rather than writing to the standard output (stdout), the result is placed in a null terminated string pointed to by the first argument (which is assumed to be of type pointer to character). The parameters to sprintf consist of a pointer to char, followed by a pointer to a null terminated string, followed by zero or more arguments. Sprintf formats the arguments following the control string, using the control string to direct formatting and conversion. It places the result in the string pointed to by the first argument which must be long enough to accept it. The control string may contain normal characters (which are simply copied to the output string) and conversion specifications which control the cooying of the arguments. Each conversion specification provides information used to format its corresponding argument following the control string. Conversion specifications begin with a percent character, (%), perhaps followed by some options, and terminated by a conversion character. All the options are, of course, optional but those that are included must appear in the specified order. The legal options (in the order they must appear) are as follows:

Dash (-): indicates that, if the number to be copied is shorter than the specified field length, it should be left justified. if this option is omitted the number will be right justified.

Zero (0): indicates that, if the number to be copied is shorter than the specified field length, it should be padded with zeros to fill th field length. If this option is omitted the field will be padded with blanks.

Digit string: indicates the minimum field width. The argument will be copied into a field at least this wide. This field may be replaced with a star (*) which will cause the field width to be taken from the next corresponding argument (assumed an integer) in the argument list.

Period (.): separates the field width from the next digit string.

Digit string: indicates the precision. For a float the precision is the number of digits to be written to the right of the decimal point. For a string the precision is the maximum number of characters which will be written. This field may be replaced with a star (*) which will cause the field width to be taken from the next

corresponding argument (assumed to be an integer) in the argument list

Long (l): (letter ell) indicates that its corresponding argument is to be written as a long rather than an int.

The valid conversion characters and their meanings are as follows:

d The argument is assumed to be of type int and is written in decimal notation.

o The argument is written in octal (without leading 0).

x Argument is written in hexadecimal (without leading Ox).

u The argument is assumed to be unsigned and written in decimal notation.

c The argument is written as a character.

s The argument is assumed to be a pointer to a null terminated string. Characters are copied from the control string to the output string until a null character is reached or until the number of characters given by the precision are copied. The terminating null is not copied.

e The argument is assumed to be a float and written out in a decimal notation of the following form:
[-]d.dddddde[+|-]dd That is a negative sign if the number is negative, a single digit, followed by a decimal point, followed by several digits, followed by an 'e', followed by a sign, followed by two digits.

f The argument is assumed to be a float and written out in a decimal notation of the following form: [-]ddd.dddd where the length of the string of digits following the decimal point is given by the precision.

g Prints in either e or f format; whichever is shorter.

if a character which is neither an option nor a conversion character is found while scanning a conversion specification the character following the percent sign (%) is simply written and no conversion specification is assumed. Thus to write a percent sign one writes it twice (%%)

DIAGNOSTICS

SEE ALSO

printf(), fprintf()

NOTES

NAME

sscanf - formatted string conversion

SYNOPSIS

```
int     sscanf(string,control [,pointer1] ... )
char    *string, *control;
```

DESCRIPTION

Sscanf is nearly identical to fscanf except that its input is taken from the string pointed to by the first argument rather than a file. The parameters to sscanf consist of a pointer to char, followed by a pointer to a null terminated string (the control string), followed by zero or more arguments of type pointer. Sscanf reads groups of characters from the input string pointed to by the first argument, interprets them according to the control string, and writes the results into the arguments pointed to by their corresponding argument pointers. The control string may contain blanks, tabs, and newlines which match optional white space in the input string; it may contain ordinary characters which must match the input string exactly character per character; and it may contain conversion specifications used to control the interpretation of the input string. Each conversion specification provides information used to translate a segment of the input string into a value which may then be placed into an argument pointed to by its corresponding pointer in the argument list.

Conversion specifications begin with a percent character, (%), perhaps followed by some options, and terminated by a conversion character. All the options are, of course, optional but those that are included must appear in the specified order.

The legal options (in the order they must appear) are as follows:

Star (*) indicates that this conversion specification has no corresponding pointer in the argument list. This effectively skips a value in the input string.

Digit string: indicates the maximum field width; the maximum number of characters which this conversion specification will cause to be read off the input string.

Long (l): (letter ell) indicates that the corresponding pointer is pointing to a long rather than an int. This has no effect when preceding an e or f.

The valid conversion characters and their meanings are as follows:

d A decimal integer is expected in the input string. Its corresponding pointer is assumed to be of type lint.

o An octal integer is expected in the input string. Its corresponding pointer is assumed to be of type *int.

x A hexadecimal integer is expected in the input string. Its corresponding pointer is assumed to be of type *int.

h A decimal integer is expected in the input string. Its corresponding pointer is assumed to be of type *short.

u An unsigned decimal integer is expected in the input string. Its corresponding pointer is assumed to be of type *unsigned.

c The very next character is read from the input string (even if it's a blank). Its corresponding pointer is assumed to be of type *char.

S A string is expected in the input string. Its corresponding pointer is assumed to be of type *char. It should point to a space large enough to hold the input string plus an added null. Characters are read, starting with the next nonblank character, until the number of characters given in the precision is reached or until a blank, tab, or newline is reached.

e (same as f)

f A floating point number is expected in the input string. Its corresponding pointer is assumed to be of type *float.

The return value of this function is the number of parameters that were matched (read in off the input line) or EOF.

DIAGNOSTICS

SEE ALSO

scanf(), fscanf()

NOTES

A hexadecimal number may not be preceded by a Ox.

Any character within a conversion specifier which is not a legal conversion specifier option or conversion character will be ignored along with the preceding percent sign and any characters inbetween. Thus there is no way to match a '%' on the input line (i.e. writings %% in the control string will not cause it to try to match a % in the input string).

NAME

strcat - copy string

SYNOPSIS

```
int     strcat(sl,s2)
char    *sl,*s2;
```

DESCRIPTION

Strcat appends a copy of the string pointed to by its second argument to the end of the string pointed to by its first argument. It is assumed that the first argument points to an area large enough to accomodate the resultant string.

DIAGNOSTICS

SEE ALSO

strcmp(), strlen(), strsave()

NOTES

NAME

strcmp - compare strings lexicographically

SYNOPSIS

```
int     strcmp(sl,s2)
char    *sl,*s2;
```

DESCRIPTION

Strcmp lexicographically compares its first argument with its second. It returns 1 if the first is greater than the second, 0 if the two are equal, and -1 if the first is less than the second.

DIAGNOSTICS

SEE ALSO

strcpy(), strlen(), strsave()

NOTES

NAME

strcpy - copy string

SYNOPSIS

```
int     strcpy(sl,s2)
char    *sl,*s2;
```

DESCRIPTION

Strcpy copies the string pointed to by the second argument to the area pointed to by the first. It stops after a null character has been conied.

DIAGNOSTICS

SEE ALSO

strcmp(), strlen(), strsaveo

NOTES

NAME

strlen - return string length

SYNOPSIS

```
int     strlen(s)
char    *s;
```

DESCRIPTION

Strlen returns the length of the string pointed to by the argument (not including the terminating null).

DIAGNOSTICS

SEE ALSO

strcmp(), stcpy(), strsave()

NOTES

NAME

strncat - copy string

SYNOPSIS

```
int      strncat(sl,s2,n)
char     *sl,*s2;
int      n;
```

DESCRIPTION

Strncat appends a copy of the string pointed to by its second argument to the end of the string pointed to by its first argument. Strncat copies at most the number of characters specified by its third argument. It is assumed that the first argument points to an area large enough to accomodate the resultant string.

DIAGNOSTICS

SEE ALSO

strcat(), strcmd(), strlen(), strsave()

NOTES

NAME

strncmp - compare strings lexicographically

SYNOPSIS

```
int      strncmp(sl,s2,n)
char     *sl,*s2;
int      n;
```

DESCRIPTION

Strncmp lexicographically compares its first argument with its second. It returns 1 if the first is greater than the second, 0 if the two are equal, and -1 if the first is less than the second. Strncmp compares at most the number of characters specified by its third argument; any others are not considered.

DIAGNOSTICS

SEE ALSO

strcmp(), strcpy(), strlen(), strsave()

NOTES

NAME

strncpy - copy string

SYNOPSIS

```
int     strncpy (s1,s2,n)
char    *sl,*s2;
int n;
```

DESCRIPTION

Strncpy copies the string pointed to by the second argument to the area pointed to by the first. It stops after it has copied the number of characters specified by its third argument or when a null character has been copied.

DIAGNOSTICS

SEE ALSO

strcmp(), strcpy(), strlen(), strsave()

NOTES

NAME

strsave - save string in memory

SYNOPSIS

```
char    *strsave(s)
char    *S;
```

DESCRIPTION

Strsave attempts to allocate a space in memory large enough to hold the string pointed to by the argument (plus its terminating null). If it succeeds strsave copies the string pointed to by the argument into the memory and returns a pointer to it. If it fails to allocate sufficient memory, strsave returns NULL.

The area used by "strsave" to save the string is obtained by a call to "alloc" and thus may be returned to the system by a call to "free" using the string pointer as an argument.

DIAGNOSTICS

SEE ALSO

alloc(), free(), strcmp(), strcpy(), strlen()

NOTES

NAME

tolower - convert to lower case

SYNOPSIS

```
char    tolower(ch)
char    ch;
```

DESCRIPTION

Returns its argument converted to lower case

DIAGNOSTICS

SEE ALSO

toupper()

NOTES

NAME

toupper - convert to upper case

SYNOPSIS

```
char    toupper(ch)
char    ch;
```

DESCRIPTION

Returns its argument converted to upper case

DIAGNOSTICS

SEE ALSO

tolower()

NOTES

NAME

uldiv unsigned long integer divide

SYNOPSIS

```
long    uldiv(opl,op2)
long    opl,op2;
```

DESCRIPTION

Uldiv returns a long (unsigned) integer which represents the nonfractional result of dividing the first (unsigned) long integer argument by the second (unsigned) long integer argument.

DIAGNOSTICS

Division by 0 will return (long) -1.

SEE ALSO

ulmod(), ulmul()

NOTES

There is actually no type "unsigned long". Uldiv operates on longs as if they were unsigned by ignoring the normal sign conventions.

NAME

ulmod - unsigned long modulo operation

SYNOPSIS

```
long    ulmod (opl, op2)
long    opl,op2;
```

DESCRIPTION

Ulmod returns a long (unsigned) integer which represents the remainder of the result produced by dividing the first (unsigned) long integer argument by the second (unsigned) long integer argument.

DIAGNOSTICS

When the second argument is zero (division by 0) the function returns the first argument.

SEE ALSO

uldiv(), ulmul()

NOTES

There is actually no type "unsigned long". Ulmod operates on longs as if they were unsigned by ignoring the normal sign conventions.

NAME

ulmul - unsigned long multiply

SYNOPSIS

```
long    ulmul (opl, op2)
long    opl,op2;
```

DESCRIPTION

Ulmul returns a long (unsigned) integer which represents the result of multiplying the first (unsigned) long integer argument by the second (unsigned) long integer argument.

DIAGNOSTICS

SEE ALSO

uldiv(), ulmod

NOTES

There is actually no type "unsigned long". Ulmul operates on longs as if they were unsigned by ignoring the normal sign conventions.

NAME

_unext - unextend float

SYNOPSIS

```
float    unext(ef)
struct   extflt
         {
         char    sign;
         int     exp;
         long    mantissa;
         } *ef;
```

DESCRIPTION

_unext returns the float which is represented by the extended floating point number contained in the structure pointed to by the argument. The first element of the structure is assumed to contain the sign bit of the number, the second element should contain the unbiased exponent, and the third the mantissa.

DIAGNOSTICS

SEE ALSO

_extend()

NOTES

NAME

ungetc - push character back on input stream

SYNOPSIS

```
#include "stdio.h"
int     ungetc (c, fp)
FILE    *fp;
int     c;
```

DESCRIPTION

Ungetc attempts to push a character back on the input stream so that it will be the next one retrieved. At most one character may be pushed back inbetween calls to getc. The first argument is the character to be pushed the second is a pointer to the file into which the character is to be pushed. The file pointer must have been previously returned from an fopen call unless it is STDIN.

DIAGNOSTICS

Ungetc returns ERROR (-1) if it could not push the character.

SEE ALSO

getc()

NOTES

NAME

ungetchar - push character back on standard innut stream

SYNOPSIS

```
#include "stdio.h"
int      ungetchar(c)
char     c;
```

DESCRIPTION

Ungetchar attempts to push a character back on the standard input stream so that it will be the next one retrieved. At most one character may be pushed back inbetween calls to getchar. The argument is the character to be pushed. This call is equivalent to ungetc (c, STDIN)

DIAGNOSTICS

Ungetchar returns ERROR (-1) if it could not push the character.

SEE ALSO

NOTES

NAME

unlink - delete file

SYNOPSIS

```
int     unlink(name)
char    *name;
```

DESCRIPTION

Unlink deletes the file whose name is contained in the string pointed to by its argument. Under some operating systems unlink simply decreases a link count to the file and deletes the file if the link count reaches zero as a result.

DIAGNOSTICS

Unlink returns ERROR if the file could not be cveleted.

SEE ALSO

NOTES

Under the Flex and 0S9 operating systems unlink simply has the effect of deleting the file. Under more Unix like operating systems such as UniFLEX unlink decreases the link count on the file. Such an operating system will delete any file whose link count decreases to zero. There is a companion library routine, link(), which increases the link count on a file for those operating systems which support it.

ADDENDUM TO THE INTROL-C USER MANUAL

LINKER AND LOADER REFERENCE MANUAL

-b Option

Two forms of the' "-b" option described on page L.1.6 of the Linker And Loader Reference Manual are now available:

-b -or- -b=<Pathnarne>

The first form above, "-b", prevents the Standard Library, libc.R, from being searched by the Linker. The second form, "-b=<Pathname>", defines <pathname> as being a non-standard place in which to find the Standard Library, libc.R.

-i Option

A "-i" option has been added for the Linker. When. a -i is specified on the link command line, this option specifier will force loading of all modules on the command line.

-l Option

Two forms of the "-l" option described on page L.1.8. of the Linker And Loader Reference Manual are now available:

-l[s][x][u][=<file>] -or- -ll[s][x][u][=<file>]

The first form above, where a single leading "1" is specified, causes a linker listing to be produced exactly as described on page L.1.8 of the User Manual. The second form, where a double leading "l" is used, instead causes a loader listing to be produced. That is, an option specification beginning with "-l" will be ignored by the linker itself and passed intact to the loader to cause a loader listing to be generated.

-r option

A "-r" option has been added for the Linker. The -r option specifier causes the .RL output file generated by the Linker to be saved during an automatic link-and-load sequence. Normally (when the -r option is not specified), when the Linker automatically calls the Loader, the Linker passes the Loader a "-z" option specifier which causes the Loader to delete its input file (ie the Linker's .RL output file) when the Loader has finished with it. Specifying the -r option on the link command line inhibits the Linker from passing the -z specifier to the Loader, thus causing the intermediate RL Linker output file to be retained.

STANDARD LIBRARY REFERENCE MANUAL (UC6809 Library Only)

The Standard Library Reference Manual erroneously describes two routines that do not exist in the supplied Standard Library:

rand - Return random number
srand - Set seed for random number generator

Therefore, please delete/ignore the descriptions for these two routines.

APPENDIX A
FC6809 STANDARD LIBRARY

NON-ZERO CLASS LIBRARY ROUTINES

As discussed in the Compiler Reference manual and Linker Reference manual, all relocatable modules (including those contained in the Standard Library) have a special identifying attribute called a "class" specifier, which is a number in the range 0 through 255. At link time, the Linker uses a module's class number to differentiate between different versions of identically named modules that may possibly co-exist within the same library.

In the case of the FC6809 Standard Library, most of the function modules supplied in the library have a preassigned modure class specifier of "O" (zero). In fact, each of the various runtime support functions is furnished and available for use as a class 0 type of module. However, the library also includes "alternate" versions of some runtime functions. Where such alternate support routines exist, they have been given the same filename as the "standard" version of the routine, but have been assigned non-zero class numbers.

In all cases, the class 0 version of a given library routine will always provide the full runtime support features that have been described for that routine in this reference manual. Any non-zero classes of library routines, by comparison, provide a modified (and typically abbreviated) level of support for the given runtime function, usually resulting in smaller runtime overhead in the final program.

Four non-zero class categories of library functions are included in the FC6809 Standard Library; class 5, class 6, class 7, and class 8.

Classes 5 and 6 are associated with selection of modified versions of the output formatting routines, such as printf, fprintf, and sprintf; classes 7 and 8 select modified versions of the input formatting routines, such as scanf, fscanf, and sscanf. Whereas the class 0 versions of these respective routines provide full support for longs, integers, and floating point numbers, the non-zero class versions differ as follows:

Class 5 - Output formatting routines will support only integers.

Class 6 - Output formatting routines will support only integers and longs.

Class 7 - Input formatting routines will support only integers.

Class 8 - Input formatting routines will support only integers and longs.

APPENDIX D

INSTALLATION OF THE FC6809 INTROL-C COMPILER

This section describes the installation of Introl-C on the Flex operating system.

The FC6809 Introl-C Compiler is shipped on standard 8 inch or 5 inch floppy disk format. Verify that the disk is indeed intended for the Flex operating system and also that the disk format is what you expect by reading the label on the distribution diskette envelope. Note that the disk shipped to you is not bootable and thus cannot be used to start your Flex system.

Before it can be used, the Compiler and its associated programs must be copied from the distribution disk to the system drive. Unless specified otherwise, the program to be compiled is assumed to be on the work drive.

Notice that the "stdio.h", "flex.h", and "setjmp.h" files are NOT capitalized. When you copy these files, be sure that their names are in lower case. On many FLEX systems file names are automatically converted to upper case even when typed in lower case. Many systems already have a utility to defeat this "feature" but, if not, the distribution disk includes a utivity called "CASE" which, when run, prevents this automatic conversion. The CASE program toggles between 'upper/lower case' and 'upper case only' each time it is run so if it is run an even number of times the system will again convert lower case to upper.

You may also wish to take note of the other files you find on your distribution disk. They include source code examples of many of the standard library routines and perhaps some useful or interesting routines. See your FLEX System Users Manual for details on making copies of files.