NAME

fopen, fdopen, fileno – stream open functions

SYNOPSIS

#include <stdio.h>

FILE *fopen(const char * *path***, const char *** *mode***);**
FILE *fdopen(int *fildes***, const char *** *mode***);**
int fileno(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.

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 appending (writing at end of file). The file is created if it does not exist. The stream is positioned at the end of the file.

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

The **fdopen** function associates a stream with the existing file descriptor, *fildes*. The *mode* of the stream (one of the values "r", "r+", "w", "w+", "a", "a+") must be compatible with the mode of the file descriptor. The file position indicator of the new stream is set to that belonging to *fildes*, and the error and end-of-file indicators are cleared. Modes "w" or "w+" do not cause truncation of the file. The file descriptor is not dup'ed, and will be closed when the stream created by **fdopen** is closed. The result of applying **fdopen** to a shared memory object is undefined.

The function **fileno**() examines the argument *stream* and returns its integer descriptor.

RETURN VALUE

Upon successful completion **fopen**, **fdopen** 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**, **fdopen**, or **freopen** was invalid.

The **fopen**, **fdopen** and **freopen** functions may also fail and set *errno* for any of the errors specified for the routine **malloc**(3).

The **fopen** function may also fail and set *errno* for any of the errors specified for the routine **open**(2).

The **fdopen** function may also fail and set *errno* for any of the errors specified for the routine **fcntl**(2).

SEE ALSO

open(2), **fclose**(3), **fileno**(3)

NAME

fgetc, fgets, getc, getchar, fputc, fputs, putc, putchar – input and output of characters and strings

SYNOPSIS

#include <stdio.h>

int fgetc(FILE * *stream***);**
char *fgets(char * *s***, int** *size***, FILE *** *stream***);**
int getc(FILE * *stream***);**
int getchar(void);
int fputc(int *c***, FILE *** *stream***);**
int fputs(const char * *s***, FILE *** *stream***);**
int putc(int *c***, FILE *** *stream***);**
int putchar(int *c***);**

DESCRIPTION

fgetc() reads the next character from *stream* and returns it as an *unsigned char* cast to an *int*, or **EOF** on end of file or error.

getc() is equivalent to **fgetc**() except that it may be implemented as a macro which evaluates *stream* more than once.

getchar() is equivalent to **getc**(*stdin*).

fgets() reads in at most one less than *size* characters from *stream* and stores them into the buffer pointed to by *s*. Reading stops after an **EOF** or a newline. If a newline is read, it is stored into the buffer. A **'\0'** is stored after the last character in the buffer.

fputc() writes the character *c*, cast to an *unsigned char*, to *stream*.

fputs() writes the string *s* to *stream*, without its terminating null byte ('\0').

putc() is equivalent to **fputc**() except that it may be implemented as a macro which evaluates *stream* more than once.

putchar(*c*); is equivalent to **putc**(*c*, *stdout*).

Calls to the functions described here can be mixed with each other and with calls to other output functions from the *stdio* library for the same output stream.

RETURN VALUE

fgetc(), **getc**() and **getchar**() return the character read as an *unsigned char* cast to an *int* or **EOF** on end of file or error.

fgets() returns *s* on success, and NULL on error or when end of file occurs while no characters have been read. **fputc**(), **putc**() and **putchar**() return the character written as an *unsigned char* cast to an *int* or **EOF** on error.

fputs() returns a nonnegative number on success, or **EOF** on error.

SEE ALSO

read(2), **write**(2), **ferror**(3), **fgetwc**(3), **fgetws**(3), **fopen**(3), **fread**(3), **fseek**(3), **getline**(3), **getwchar**(3), **scanf**(3), **ungetwc**(3), **write**(2), **ferror**(3), **fopen**(3), **fputwc**(3), **fputws**(3), **fseek**(3), **fwrite**(3), **gets**(3), **putwchar**(3), **scanf**(3), **unlocked_stdio**(3)

NAME

calloc, malloc, free, realloc – Allocate and free dynamic memory

SYNOPSIS

```
#include <stdlib.h>

void *calloc(size_t nmemb, size_t size);
void *malloc(size_t size);
void free(void *ptr);
void *realloc(void *ptr, size_t size);
```

DESCRIPTION

calloc() allocates memory for an array of *nmemb* elements of *size* bytes each and returns a pointer to the allocated memory. The memory is set to zero.

malloc() allocates *size* bytes and returns a pointer to the allocated memory. The memory is not cleared.

free() frees the memory space pointed to by *ptr*, which must have been returned by a previous call to **malloc()**, **calloc()** or **realloc()**. Otherwise, or if **free(***ptr***)** has already been called before, undefined behaviour occurs. If *ptr* is **NULL**, no operation is performed.

realloc() changes the size of the memory block pointed to by *ptr* to *size* bytes. The contents will be unchanged to the minimum of the old and new sizes; newly allocated memory will be uninitialized. If *ptr* is **NULL**, the call is equivalent to **malloc(size)**; if size is equal to zero, the call is equivalent to **free(***ptr***)**. Unless *ptr* is **NULL**, it must have been returned by an earlier call to **malloc()**, **calloc()** or **realloc()**.

RETURN VALUE

For **calloc()** and **malloc()**, the value returned is a pointer to the allocated memory, which is suitably aligned for any kind of variable, or **NULL** if the request fails.

free() returns no value.

realloc() returns a pointer to the newly allocated memory, which is suitably aligned for any kind of variable and may be different from *ptr*, or **NULL** if the request fails. If *size* was equal to 0, either NULL or a pointer suitable to be passed to *free*() is returned. If **realloc()** fails the original block is left untouched - it is not freed or moved.

CONFORMING TO

ANSI-C

SEE ALSO

brk(2), **posix_memalign**(3)

NAME

qsort – sorts an array

SYNOPSIS

```
#include <stdlib.h>

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

DESCRIPTION

The **qsort**() function sorts an array with *nmemb* elements of size *size*. The *base* argument points to the start of the array.

The contents of the array are sorted in ascending order according to a comparison function pointed to by *compar*, which is called with two arguments that point 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. If two members compare as equal, their order in the sorted array is undefined.

RETURN VALUE

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

SEE ALSO

sort(1), **alphasort**(3), **strcmp**(3), **versionsort**(3)

ATTRIBUTES

Multithreading (see pthreads(7))

The **qsort**() function is thread-safe if the comparison function *compar* does not access any global variables.

COLOPHON

This page is part of release 3.05 of the Linux *man-pages* project. A description of the project, and information about reporting bugs, can be found at http://www.kernel.org/doc/man-pages/.

NAME

strcmp, strncmp – compare two strings

SYNOPSIS

#include <string.h>

int strcmp(const char ****s1*, const char ****s2***);**

int strncmp(const char ****s1*, const char ****s2***, size_t** *n***);**

DESCRIPTION

The **strcmp**() function compares the two strings *s1* and *s2*. It returns an integer less than, equal to, or greater than zero if *s1* is found, respectively, to be less than, to match, or be greater than *s2*.

The **strncmp**() function is similar, except it only compares the first (at most) *n* characters of *s1* and *s2*.

RETURN VALUE

The **strcmp**() and **strncmp**() functions return an integer less than, equal to, or greater than zero if *s1* (or the first *n* bytes thereof) is found, respectively, to be less than, to match, or be greater than *s2*.

CONFORMING TO

SVr4, 4.3BSD, C89, C99.

SEE ALSO

bcmp(3), **memcmp**(3), **strcasecmp**(3), **strcoll**(3), **strncasecmp**(3), **wcscmp**(3), **wcsncmp**(3)

NAME

strdup, strndup – duplicate a string

SYNOPSIS

#include <string.h>

char *strdup(const char ****s*);**

char *strndup(const char ****s*, size_t** *n***);**

DESCRIPTION

The **strdup**() function returns a pointer to a new string which is a duplicate of the string *s*. Memory for the new string is obtained with **malloc**(3), and can be freed with **free**(3).

The **strndup**() function is similar, but copies at most *n* bytes. If *s* is longer than *n*, only *n* bytes are copied, and a terminating null byte ('\0') is added.

RETURN VALUE

On success, the **strdup**() function returns a pointer to the duplicated string. It returns NULL if insufficient memory was available, with *errno* set to indicate the cause of the error.

ERRORS

ENOMEM

Insufficient memory available to allocate duplicate string.

CONFORMING TO

strdup() conforms to SVr4, 4.3BSD, POSIX.1-2001. **strndup**() conforms to POSIX.1-2008.