```
/* Define ISO C stdio on top of C++ iostreams.
   Copyright (C) 1991, 1994-2007, 2008 Free Software Foundation, Inc.
   This file is part of the GNU C Library.

   The GNU C Library is free software; you can redistribute it and/or
   modify it under the terms of the GNU Lesser General Public
   License as published by the Free Software Foundation; either
   version 2.1 of the License, or (at your option) any later version.

   The GNU C Library is distributed in the hope that it will be useful,
   but WITHOUT ANY WARRANTY; without even the implied warranty of
   MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the GNU
   Lesser General Public License for more details.

   You should have received a copy of the GNU Lesser General Public
   License along with the GNU C Library; if not, write to the Free
   Software Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA
   02111-1307 USA.  */

/*
 *	ISO C99 Standard: 7.19 Input/output	<stdio.h>
 */

#ifndef _STDIO_H

#if !defined __need_FILE && !defined __need___FILE
# define _STDIO_H	1
# include <features.h>

__BEGIN_DECLS

# define __need_size_t
# define __need_NULL
# include <stddef.h>

# include <bits/types.h>
# define __need_FILE
# define __need___FILE
#endif /* Don't need FILE.  */


#if !defined __FILE_defined && defined __need_FILE

/* Define outside of namespace so the C++ is happy.  */
struct _IO_FILE;

__BEGIN_NAMESPACE_STD
/* The opaque type of streams.  This is the definition used elsewhere.  */
typedef struct _IO_FILE FILE;
__END_NAMESPACE_STD
#if defined __USE_LARGEFILE64 || defined __USE_SVID || defined __USE_POSIX \
    || defined __USE_BSD || defined __USE_ISOC99 || defined __USE_XOPEN \
    || defined __USE_POSIX2
__USING_NAMESPACE_STD(FILE)
#endif

# define __FILE_defined  1
#endif /* FILE not defined.  */
```

```
#undef   __need_FILE


#if !defined ____FILE_defined && defined __need___FILE

/* The opaque type of streams.  This is the definition used elsewhere.  */
typedef struct _IO_FILE __FILE;

# define ____FILE_defined          1
#endif /* __FILE not defined.  */
#undef   __need___FILE


#ifdef    _STDIO_H
#define _STDIO_USES_IOSTREAM

#include <libio.h>

#ifdef __USE_XOPEN
# ifdef __GNUC__
#  ifndef _VA_LIST_DEFINED
typedef _G_va_list va_list;
#   define _VA_LIST_DEFINED
#  endif
# else
#  include <stdarg.h>
# endif
#endif

/* The type of the second argument to `fgetpos' and `fsetpos'.  */
__BEGIN_NAMESPACE_STD
#ifndef __USE_FILE_OFFSET64
typedef _G_fpos_t fpos_t;
#else
typedef _G_fpos64_t fpos_t;
#endif
__END_NAMESPACE_STD
#ifdef __USE_LARGEFILE64
typedef _G_fpos64_t fpos64_t;
#endif

/* The possibilities for the third argument to `setvbuf'.  */
#define _IOFBF 0                        /* Fully buffered.  */
#define _IOLBF 1                        /* Line buffered.  */
#define _IONBF 2                        /* No buffering.  */


/* Default buffer size.  */
#ifndef BUFSIZ
# define BUFSIZ _IO_BUFSIZ
#endif


/* End of file character.
   Some things throughout the library rely on this being -1.  */
#ifndef EOF
# define EOF (-1)
#endif
```

```
/* The possibilities for the third argument to `fseek'.
   These values should not be changed.  */
#define SEEK_SET        0       /* Seek from beginning of file.  */
#define SEEK_CUR        1       /* Seek from current position.  */
#define SEEK_END        2       /* Seek from end of file.  */


#if defined __USE_SVID || defined __USE_XOPEN
/* Default path prefix for `tempnam' and `tmpnam'.  */
# define P_tmpdir               "/tmp"
#endif


/* Get the values:
   L_tmpnam     How long an array of chars must be to be passed to `tmpnam'.
   TMP_MAX      The minimum number of unique filenames generated by tmpnam
                (and tempnam when it uses tmpnam's name space),
                or tempnam (the two are separate).
   L_ctermid    How long an array to pass to `ctermid'.
   L_cuserid    How long an array to pass to `cuserid'.
   FOPEN_MAX    Minimum number of files that can be open at once.
   FILENAME_MAX Maximum length of a filename.  */
#include <bits/stdio_lim.h>


/* Standard streams.  */
extern struct _IO_FILE *stdin;          /* Standard input stream.  */
extern struct _IO_FILE *stdout;         /* Standard output stream.  */
extern struct _IO_FILE *stderr;         /* Standard error output stream.  */
/* C89/C99 say they're macros.  Make them happy.  */
#define stdin stdin
#define stdout stdout
#define stderr stderr

__BEGIN_NAMESPACE_STD
/* Remove file FILENAME.  */
extern int remove (__const char *__filename) __THROW;
/* Rename file OLD to NEW.  */
extern int rename (__const char *__old, __const char *__new) __THROW;
__END_NAMESPACE_STD

#ifdef __USE_ATFILE
/* Rename file OLD relative to OLDFD to NEW relative to NEWFD.  */
extern int renameat (int __oldfd, __const char *__old, int __newfd,
                     __const char *__new) __THROW;
#endif

__BEGIN_NAMESPACE_STD
/* Create a temporary file and open it read/write.

   This function is a possible cancellation points and therefore not
   marked with __THROW.  */
#ifndef __USE_FILE_OFFSET64
extern FILE *tmpfile (void) __wur;
#else
# ifdef __REDIRECT
```

```
extern FILE *__REDIRECT (tmpfile, (void), tmpfile64) __wur;
# else
#  define tmpfile tmpfile64
# endif
#endif

#ifdef __USE_LARGEFILE64
extern FILE *tmpfile64 (void) __wur;
#endif

/* Generate a temporary filename.  */
extern char *tmpnam (char *__s) __THROW __wur;
__END_NAMESPACE_STD

#ifdef __USE_MISC
/* This is the reentrant variant of `tmpnam'.  The only difference is
   that it does not allow S to be NULL.  */
extern char *tmpnam_r (char *__s) __THROW __wur;
#endif


#if defined __USE_SVID || defined __USE_XOPEN
/* Generate a unique temporary filename using up to five characters of PFX
   if it is not NULL.  The directory to put this file in is searched for
   as follows: First the environment variable "TMPDIR" is checked.
   If it contains the name of a writable directory, that directory is used.
   If not and if DIR is not NULL, that value is checked.  If that fails,
   P_tmpdir is tried and finally "/tmp".  The storage for the filename
   is allocated by `malloc'.  */
extern char *tempnam (__const char *__dir, __const char *__pfx)
     __THROW __attribute_malloc__ __wur;
#endif


__BEGIN_NAMESPACE_STD
/* Close STREAM.

   This function is a possible cancellation point and therefore not
   marked with __THROW.  */
extern int fclose (FILE *__stream);
/* Flush STREAM, or all streams if STREAM is NULL.

   This function is a possible cancellation point and therefore not
   marked with __THROW.  */
extern int fflush (FILE *__stream);
__END_NAMESPACE_STD

#ifdef __USE_MISC
/* Faster versions when locking is not required.

   This function is not part of POSIX and therefore no official
   cancellation point.  But due to similarity with an POSIX interface
   or due to the implementation it is a cancellation point and
   therefore not marked with __THROW.  */
extern int fflush_unlocked (FILE *__stream);
#endif

#ifdef __USE_GNU
```

```
/* Close all streams.

   This function is not part of POSIX and therefore no official
   cancellation point.  But due to similarity with an POSIX interface
   or due to the implementation it is a cancellation point and
   therefore not marked with __THROW.  */
extern int fcloseall (void);
#endif


__BEGIN_NAMESPACE_STD
#ifndef __USE_FILE_OFFSET64
/* Open a file and create a new stream for it.

   This function is a possible cancellation point and therefore not
   marked with __THROW.  */
extern FILE *fopen (__const char *__restrict __filename,
                    __const char *__restrict __modes) __wur;
/* Open a file, replacing an existing stream with it.

   This function is a possible cancellation point and therefore not
   marked with __THROW.  */
extern FILE *freopen (__const char *__restrict __filename,
                      __const char *__restrict __modes,
                      FILE *__restrict __stream) __wur;
#else
# ifdef __REDIRECT
extern FILE *__REDIRECT (fopen, (__const char *__restrict __filename,
                                 __const char *__restrict __modes), fopen64)
  __wur;
extern FILE *__REDIRECT (freopen, (__const char *__restrict __filename,
                                   __const char *__restrict __modes,
                                   FILE *__restrict __stream), freopen64)
  __wur;
# else
#  define fopen fopen64
#  define freopen freopen64
# endif
#endif
__END_NAMESPACE_STD
#ifdef __USE_LARGEFILE64
extern FILE *fopen64 (__const char *__restrict __filename,
                      __const char *__restrict __modes) __wur;
extern FILE *freopen64 (__const char *__restrict __filename,
                        __const char *__restrict __modes,
                        FILE *__restrict __stream) __wur;
#endif

#ifdef    __USE_POSIX
/* Create a new stream that refers to an existing system file descriptor.  */
extern FILE *fdopen (int __fd, __const char *__modes) __THROW __wur;
#endif

#ifdef    __USE_GNU
/* Create a new stream that refers to the given magic cookie,
   and uses the given functions for input and output.  */
extern FILE *fopencookie (void *__restrict __magic_cookie,
                          __const char *__restrict __modes,
```

```
                                _IO_cookie_io_functions_t __io_funcs) __THROW __wur;

/* Create a new stream that refers to a memory buffer.  */
extern FILE *fmemopen (void *__s, size_t __len, __const char *__modes)
  __THROW __wur;

/* Open a stream that writes into a malloc'd buffer that is expanded as
   necessary.  *BUFLOC and *SIZELOC are updated with the buffer's location
   and the number of characters written on fflush or fclose.  */
extern FILE *open_memstream (char **__bufloc, size_t *__sizeloc) __THROW __wur;
#endif


__BEGIN_NAMESPACE_STD
/* If BUF is NULL, make STREAM unbuffered.
   Else make it use buffer BUF, of size BUFSIZ.  */
extern void setbuf (FILE *__restrict __stream, char *__restrict __buf) __THROW;
/* Make STREAM use buffering mode MODE.
   If BUF is not NULL, use N bytes of it for buffering;
   else allocate an internal buffer N bytes long.  */
extern int setvbuf (FILE *__restrict __stream, char *__restrict __buf,
                    int __modes, size_t __n) __THROW;
__END_NAMESPACE_STD

#ifdef     __USE_BSD
/* If BUF is NULL, make STREAM unbuffered.
   Else make it use SIZE bytes of BUF for buffering.  */
extern void setbuffer (FILE *__restrict __stream, char *__restrict __buf,
                       size_t __size) __THROW;

/* Make STREAM line-buffered.  */
extern void setlinebuf (FILE *__stream) __THROW;
#endif


__BEGIN_NAMESPACE_STD
/* Write formatted output to STREAM.

   This function is a possible cancellation point and therefore not
   marked with __THROW.  */
extern int fprintf (FILE *__restrict __stream,
                    __const char *__restrict __format, ...);
/* Write formatted output to stdout.

   This function is a possible cancellation point and therefore not
   marked with __THROW.  */
extern int printf (__const char *__restrict __format, ...);
/* Write formatted output to S.  */
extern int sprintf (char *__restrict __s,
                    __const char *__restrict __format, ...) __THROW;

/* Write formatted output to S from argument list ARG.

   This function is a possible cancellation point and therefore not
   marked with __THROW.  */
extern int vfprintf (FILE *__restrict __s, __const char *__restrict __format,
                     _G_va_list __arg);
/* Write formatted output to stdout from argument list ARG.
```

```
   This function is a possible cancellation point and therefore not
   marked with __THROW.  */
extern int vprintf (__const char *__restrict __format, _G_va_list __arg);
/* Write formatted output to S from argument list ARG.  */
extern int vsprintf (char *__restrict __s, __const char *__restrict __format,
                     _G_va_list __arg) __THROW;
__END_NAMESPACE_STD

#if defined __USE_BSD || defined __USE_ISOC99 || defined __USE_UNIX98
__BEGIN_NAMESPACE_C99
/* Maximum chars of output to write in MAXLEN.  */
extern int snprintf (char *__restrict __s, size_t __maxlen,
                     __const char *__restrict __format, ...)
     __THROW __attribute__ ((__format__ (__printf__, 3, 4)));

extern int vsnprintf (char *__restrict __s, size_t __maxlen,
                      __const char *__restrict __format, _G_va_list __arg)
     __THROW __attribute__ ((__format__ (__printf__, 3, 0)));
__END_NAMESPACE_C99
#endif

#ifdef __USE_GNU
/* Write formatted output to a string dynamically allocated with `malloc'.
   Store the address of the string in *PTR.  */
extern int vasprintf (char **__restrict __ptr, __const char *__restrict __f,
                      _G_va_list __arg)
     __THROW __attribute__ ((__format__ (__printf__, 2, 0))) __wur;
extern int __asprintf (char **__restrict __ptr,
                       __const char *__restrict __fmt, ...)
     __THROW __attribute__ ((__format__ (__printf__, 2, 3))) __wur;
extern int asprintf (char **__restrict __ptr,
                     __const char *__restrict __fmt, ...)
     __THROW __attribute__ ((__format__ (__printf__, 2, 3))) __wur;

/* Write formatted output to a file descriptor.

   These functions are not part of POSIX and therefore no official
   cancellation point.  But due to similarity with an POSIX interface
   or due to the implementation they are cancellation points and
   therefore not marked with __THROW.  */
extern int vdprintf (int __fd, __const char *__restrict __fmt,
                     _G_va_list __arg)
     __attribute__ ((__format__ (__printf__, 2, 0)));
extern int dprintf (int __fd, __const char *__restrict __fmt, ...)
     __attribute__ ((__format__ (__printf__, 2, 3)));
#endif


__BEGIN_NAMESPACE_STD
/* Read formatted input from STREAM.

   This function is a possible cancellation point and therefore not
   marked with __THROW.  */
extern int fscanf (FILE *__restrict __stream,
                   __const char *__restrict __format, ...) __wur;
/* Read formatted input from stdin.
```

```
   This function is a possible cancellation point and therefore not
   marked with __THROW.  */
extern int scanf (__const char *__restrict __format, ...) __wur;
/* Read formatted input from S.  */
extern int sscanf (__const char *__restrict __s,
                   __const char *__restrict __format, ...) __THROW;

#if defined __USE_ISOC99 && !defined __USE_GNU \
    && (!defined __LDBL_COMPAT || !defined __REDIRECT) \
    && (defined __STRICT_ANSI__ || defined __USE_XOPEN2K)
# ifdef __REDIRECT
/* For strict ISO C99 or POSIX compliance disallow %as, %aS and %a[
   GNU extension which conflicts with valid %a followed by letter
   s, S or [.  */
extern int __REDIRECT (fscanf, (FILE *__restrict __stream,
                                __const char *__restrict __format, ...),
                       __isoc99_fscanf) __wur;
extern int __REDIRECT (scanf, (__const char *__restrict __format, ...),
                       __isoc99_scanf) __wur;
extern int __REDIRECT (sscanf, (__const char *__restrict __s,
                                __const char *__restrict __format, ...),
                       __isoc99_sscanf) __THROW;
# else
extern int __isoc99_fscanf (FILE *__restrict __stream,
                            __const char *__restrict __format, ...) __wur;
extern int __isoc99_scanf (__const char *__restrict __format, ...) __wur;
extern int __isoc99_sscanf (__const char *__restrict __s,
                            __const char *__restrict __format, ...) __THROW;
#  define fscanf __isoc99_fscanf
#  define scanf __isoc99_scanf
#  define sscanf __isoc99_sscanf
# endif
#endif

__END_NAMESPACE_STD

#ifdef    __USE_ISOC99
__BEGIN_NAMESPACE_C99
/* Read formatted input from S into argument list ARG.

   This function is a possible cancellation point and therefore not
   marked with __THROW.  */
extern int vfscanf (FILE *__restrict __s, __const char *__restrict __format,
                    _G_va_list __arg)
     __attribute__ ((__format__ (__scanf__, 2, 0))) __wur;

/* Read formatted input from stdin into argument list ARG.

   This function is a possible cancellation point and therefore not
   marked with __THROW.  */
extern int vscanf (__const char *__restrict __format, _G_va_list __arg)
     __attribute__ ((__format__ (__scanf__, 1, 0))) __wur;

/* Read formatted input from S into argument list ARG.  */
extern int vsscanf (__const char *__restrict __s,
                    __const char *__restrict __format, _G_va_list __arg)
     __THROW __attribute__ ((__format__ (__scanf__, 2, 0)));
```

```
# if !defined __USE_GNU \
    && (!defined __LDBL_COMPAT || !defined __REDIRECT) \
    && (defined __STRICT_ANSI__ || defined __USE_XOPEN2K)
#  ifdef __REDIRECT
/* For strict ISO C99 or POSIX compliance disallow %as, %aS and %a[
   GNU extension which conflicts with valid %a followed by letter
   s, S or [.  */
extern int __REDIRECT (vfscanf,
                       (FILE *__restrict __s,
                        __const char *__restrict __format, _G_va_list __arg),
                       __isoc99_vfscanf)
    __attribute__ ((__format__ (__scanf__, 2, 0))) __wur;
extern int __REDIRECT (vscanf, (__const char *__restrict __format,
                                _G_va_list __arg), __isoc99_vscanf)
    __attribute__ ((__format__ (__scanf__, 1, 0))) __wur;
extern int __REDIRECT (vsscanf,
                       (__const char *__restrict __s,
                        __const char *__restrict __format, _G_va_list __arg),
                       __isoc99_vsscanf)
    __THROW __attribute__ ((__format__ (__scanf__, 2, 0)));
#  else
extern int __isoc99_vfscanf (FILE *__restrict __s,
                             __const char *__restrict __format,
                             _G_va_list __arg) __wur;
extern int __isoc99_vscanf (__const char *__restrict __format,
                            _G_va_list __arg) __wur;
extern int __isoc99_vsscanf (__const char *__restrict __s,
                             __const char *__restrict __format,
                             _G_va_list __arg) __THROW;
#   define vfscanf __isoc99_vfscanf
#   define vscanf __isoc99_vscanf
#   define vsscanf __isoc99_vsscanf
#  endif
# endif

__END_NAMESPACE_C99
#endif /* Use ISO C9x.  */


__BEGIN_NAMESPACE_STD
/* Read a character from STREAM.

   These functions are possible cancellation points and therefore not
   marked with __THROW.  */
extern int fgetc (FILE *__stream);
extern int getc (FILE *__stream);

/* Read a character from stdin.

   This function is a possible cancellation point and therefore not
   marked with __THROW.  */
extern int getchar (void);
__END_NAMESPACE_STD

/* The C standard explicitly says this is a macro, so we always do the
   optimization for it.  */
#define getc(_fp) _IO_getc (_fp)
```

```
#if defined __USE_POSIX || defined __USE_MISC
/* These are defined in POSIX.1:1996.

   These functions are possible cancellation points and therefore not
   marked with __THROW.  */
extern int getc_unlocked (FILE *__stream);
extern int getchar_unlocked (void);
#endif /* Use POSIX or MISC.  */

#ifdef __USE_MISC
/* Faster version when locking is not necessary.

   This function is not part of POSIX and therefore no official
   cancellation point.  But due to similarity with an POSIX interface
   or due to the implementation it is a cancellation point and
   therefore not marked with __THROW.  */
extern int fgetc_unlocked (FILE *__stream);
#endif /* Use MISC.  */


__BEGIN_NAMESPACE_STD
/* Write a character to STREAM.

   These functions are possible cancellation points and therefore not
   marked with __THROW.

   These functions is a possible cancellation point and therefore not
   marked with __THROW.  */
extern int fputc (int __c, FILE *__stream);
extern int putc (int __c, FILE *__stream);

/* Write a character to stdout.

   This function is a possible cancellation point and therefore not
   marked with __THROW.  */
extern int putchar (int __c);
__END_NAMESPACE_STD

/* The C standard explicitly says this can be a macro,
   so we always do the optimization for it.  */
#define putc(_ch, _fp) _IO_putc (_ch, _fp)

#ifdef __USE_MISC
/* Faster version when locking is not necessary.

   This function is not part of POSIX and therefore no official
   cancellation point.  But due to similarity with an POSIX interface
   or due to the implementation it is a cancellation point and
   therefore not marked with __THROW.  */
extern int fputc_unlocked (int __c, FILE *__stream);
#endif /* Use MISC.  */

#if defined __USE_POSIX || defined __USE_MISC
/* These are defined in POSIX.1:1996.

   These functions are possible cancellation points and therefore not
   marked with __THROW.  */
extern int putc_unlocked (int __c, FILE *__stream);
```

```
extern int putchar_unlocked (int __c);
#endif /* Use POSIX or MISC.  */


#if defined __USE_SVID || defined __USE_MISC \
    || (defined __USE_XOPEN && !defined __USE_XOPEN2K)
/* Get a word (int) from STREAM.  */
extern int getw (FILE *__stream);

/* Write a word (int) to STREAM.  */
extern int putw (int __w, FILE *__stream);
#endif


__BEGIN_NAMESPACE_STD
/* Get a newline-terminated string of finite length from STREAM.

   This function is a possible cancellation point and therefore not
   marked with __THROW.  */
extern char *fgets (char *__restrict __s, int __n, FILE *__restrict __stream)
     __wur;

/* Get a newline-terminated string from stdin, removing the newline.
   DO NOT USE THIS FUNCTION!!  There is no limit on how much it will read.

   This function is a possible cancellation point and therefore not
   marked with __THROW.  */
extern char *gets (char *__s) __wur;
__END_NAMESPACE_STD

#ifdef __USE_GNU
/* This function does the same as `fgets' but does not lock the stream.

   This function is not part of POSIX and therefore no official
   cancellation point.  But due to similarity with an POSIX interface
   or due to the implementation it is a cancellation point and
   therefore not marked with __THROW.  */
extern char *fgets_unlocked (char *__restrict __s, int __n,
                             FILE *__restrict __stream) __wur;
#endif


#ifdef    __USE_GNU
/* Read up to (and including) a DELIMITER from STREAM into *LINEPTR
   (and null-terminate it). *LINEPTR is a pointer returned from malloc (or
   NULL), pointing to *N characters of space.  It is realloc'd as
   necessary.  Returns the number of characters read (not including the
   null terminator), or -1 on error or EOF.

   These functions are not part of POSIX and therefore no official
   cancellation point.  But due to similarity with an POSIX interface
   or due to the implementation they are cancellation points and
   therefore not marked with __THROW.  */
extern _IO_ssize_t __getdelim (char **__restrict __lineptr,
                               size_t *__restrict __n, int __delimiter,
                               FILE *__restrict __stream) __wur;
extern _IO_ssize_t getdelim (char **__restrict __lineptr,
                             size_t *__restrict __n, int __delimiter,
```

```
                                FILE *__restrict __stream) __wur;

/* Like `getdelim', but reads up to a newline.

   This function is not part of POSIX and therefore no official
   cancellation point.  But due to similarity with an POSIX interface
   or due to the implementation it is a cancellation point and
   therefore not marked with __THROW.  */
extern _IO_ssize_t getline (char **__restrict __lineptr,
                            size_t *__restrict __n,
                            FILE *__restrict __stream) __wur;
#endif


__BEGIN_NAMESPACE_STD
/* Write a string to STREAM.

   This function is a possible cancellation points and therefore not
   marked with __THROW.  */
extern int fputs (__const char *__restrict __s, FILE *__restrict __stream);

/* Write a string, followed by a newline, to stdout.

   This function is a possible cancellation points and therefore not
   marked with __THROW.  */
extern int puts (__const char *__s);


/* Push a character back onto the input buffer of STREAM.

   This function is a possible cancellation points and therefore not
   marked with __THROW.  */
extern int ungetc (int __c, FILE *__stream);


/* Read chunks of generic data from STREAM.

   This function is a possible cancellation points and therefore not
   marked with __THROW.  */
extern size_t fread (void *__restrict __ptr, size_t __size,
                     size_t __n, FILE *__restrict __stream) __wur;
/* Write chunks of generic data to STREAM.

   This function is a possible cancellation points and therefore not
   marked with __THROW.  */
extern size_t fwrite (__const void *__restrict __ptr, size_t __size,
                      size_t __n, FILE *__restrict __s);
__END_NAMESPACE_STD

#ifdef __USE_GNU
/* This function does the same as `fputs' but does not lock the stream.

   This function is not part of POSIX and therefore no official
   cancellation point.  But due to similarity with an POSIX interface
   or due to the implementation it is a cancellation point and
   therefore not marked with __THROW.  */
extern int fputs_unlocked (__const char *__restrict __s,
                           FILE *__restrict __stream);
```

```
#endif

#ifdef __USE_MISC
/* Faster versions when locking is not necessary.

   These functions are not part of POSIX and therefore no official
   cancellation point.  But due to similarity with an POSIX interface
   or due to the implementation they are cancellation points and
   therefore not marked with __THROW.  */
extern size_t fread_unlocked (void *__restrict __ptr, size_t __size,
                              size_t __n, FILE *__restrict __stream) __wur;
extern size_t fwrite_unlocked (__const void *__restrict __ptr, size_t __size,
                               size_t __n, FILE *__restrict __stream);
#endif


__BEGIN_NAMESPACE_STD
/* Seek to a certain position on STREAM.

   This function is a possible cancellation point and therefore not
   marked with __THROW.  */
extern int fseek (FILE *__stream, long int __off, int __whence);
/* Return the current position of STREAM.

   This function is a possible cancellation point and therefore not
   marked with __THROW.  */
extern long int ftell (FILE *__stream) __wur;
/* Rewind to the beginning of STREAM.

   This function is a possible cancellation point and therefore not
   marked with __THROW.  */
extern void rewind (FILE *__stream);
__END_NAMESPACE_STD

/* The Single Unix Specification, Version 2, specifies an alternative,
   more adequate interface for the two functions above which deal with
   file offset.  `long int' is not the right type.  These definitions
   are originally defined in the Large File Support API.  */

#if defined __USE_LARGEFILE || defined __USE_XOPEN2K
# ifndef __USE_FILE_OFFSET64
/* Seek to a certain position on STREAM.

   This function is a possible cancellation point and therefore not
   marked with __THROW.  */
extern int fseeko (FILE *__stream, __off_t __off, int __whence);
/* Return the current position of STREAM.

   This function is a possible cancellation point and therefore not
   marked with __THROW.  */
extern __off_t ftello (FILE *__stream) __wur;
# else
#  ifdef __REDIRECT
extern int __REDIRECT (fseeko,
                       (FILE *__stream, __off64_t __off, int __whence),
                       fseeko64);
extern __off64_t __REDIRECT (ftello, (FILE *__stream), ftello64);
#  else
```

```
#   define fseeko fseeko64
#   define ftello ftello64
#  endif
# endif
#endif

__BEGIN_NAMESPACE_STD
#ifndef __USE_FILE_OFFSET64
/* Get STREAM's position.

   This function is a possible cancellation point and therefore not
   marked with __THROW.  */
extern int fgetpos (FILE *__restrict __stream, fpos_t *__restrict __pos);
/* Set STREAM's position.

   This function is a possible cancellation point and therefore not
   marked with __THROW.  */
extern int fsetpos (FILE *__stream, __const fpos_t *__pos);
#else
# ifdef __REDIRECT
extern int __REDIRECT (fgetpos, (FILE *__restrict __stream,
                                 fpos_t *__restrict __pos), fgetpos64);
extern int __REDIRECT (fsetpos,
                       (FILE *__stream, __const fpos_t *__pos), fsetpos64);
# else
#  define fgetpos fgetpos64
#  define fsetpos fsetpos64
# endif
#endif
__END_NAMESPACE_STD

#ifdef __USE_LARGEFILE64
extern int fseeko64 (FILE *__stream, __off64_t __off, int __whence);
extern __off64_t ftello64 (FILE *__stream) __wur;
extern int fgetpos64 (FILE *__restrict __stream, fpos64_t *__restrict __pos);
extern int fsetpos64 (FILE *__stream, __const fpos64_t *__pos);
#endif

__BEGIN_NAMESPACE_STD
/* Clear the error and EOF indicators for STREAM.  */
extern void clearerr (FILE *__stream) __THROW;
/* Return the EOF indicator for STREAM.  */
extern int feof (FILE *__stream) __THROW __wur;
/* Return the error indicator for STREAM.  */
extern int ferror (FILE *__stream) __THROW __wur;
__END_NAMESPACE_STD

#ifdef __USE_MISC
/* Faster versions when locking is not required.  */
extern void clearerr_unlocked (FILE *__stream) __THROW;
extern int feof_unlocked (FILE *__stream) __THROW __wur;
extern int ferror_unlocked (FILE *__stream) __THROW __wur;
#endif


__BEGIN_NAMESPACE_STD
/* Print a message describing the meaning of the value of errno.
```

```
   This function is a possible cancellation point and therefore not
   marked with __THROW.  */
extern void perror (__const char *__s);
__END_NAMESPACE_STD

/* Provide the declarations for `sys_errlist' and `sys_nerr' if they
   are available on this system.  Even if available, these variables
   should not be used directly.  The `strerror' function provides
   all the necessary functionality.  */
#include <bits/sys_errlist.h>


#ifdef	__USE_POSIX
/* Return the system file descriptor for STREAM.  */
extern int fileno (FILE *__stream) __THROW __wur;
#endif /* Use POSIX.  */

#ifdef __USE_MISC
/* Faster version when locking is not required.  */
extern int fileno_unlocked (FILE *__stream) __THROW __wur;
#endif


#if (defined __USE_POSIX2 || defined __USE_SVID  || defined __USE_BSD || \
     defined __USE_MISC)
/* Create a new stream connected to a pipe running the given command.

   This function is a possible cancellation point and therefore not
   marked with __THROW.  */
extern FILE *popen (__const char *__command, __const char *__modes) __wur;

/* Close a stream opened by popen and return the status of its child.

   This function is a possible cancellation point and therefore not
   marked with __THROW.  */
extern int pclose (FILE *__stream);
#endif


#ifdef	__USE_POSIX
/* Return the name of the controlling terminal.  */
extern char *ctermid (char *__s) __THROW;
#endif /* Use POSIX.  */


#ifdef __USE_XOPEN
/* Return the name of the current user.  */
extern char *cuserid (char *__s);
#endif /* Use X/Open, but not issue 6.  */


#ifdef	__USE_GNU
struct obstack;				/* See <obstack.h>.  */

/* Write formatted output to an obstack.  */
extern int obstack_printf (struct obstack *__restrict __obstack,
                           __const char *__restrict __format, ...)
     __THROW __attribute__ ((__format__ (__printf__, 2, 3)));
```

```
extern int obstack_vprintf (struct obstack *__restrict __obstack,
                            __const char *__restrict __format,
                            _G_va_list __args)
     __THROW __attribute__ ((__format__ (__printf__, 2, 0)));
#endif /* Use GNU.  */


#if defined __USE_POSIX || defined __USE_MISC
/* These are defined in POSIX.1:1996.  */

/* Acquire ownership of STREAM.  */
extern void flockfile (FILE *__stream) __THROW;

/* Try to acquire ownership of STREAM but do not block if it is not
   possible.  */
extern int ftrylockfile (FILE *__stream) __THROW __wur;

/* Relinquish the ownership granted for STREAM.  */
extern void funlockfile (FILE *__stream) __THROW;
#endif /* POSIX || misc */

#if defined __USE_XOPEN && !defined __USE_XOPEN2K && !defined __USE_GNU
/* The X/Open standard requires some functions and variables to be
   declared here which do not belong into this header.  But we have to
   follow.  In GNU mode we don't do this nonsense.  */
# define __need_getopt
# include <getopt.h>
#endif    /* X/Open, but not issue 6 and not for GNU.  */

/* If we are compiling with optimizing read this file.  It contains
   several optimizing inline functions and macros.  */
#ifdef __USE_EXTERN_INLINES
# include <bits/stdio.h>
#endif
#if __USE_FORTIFY_LEVEL > 0 && defined __extern_always_inline
# include <bits/stdio2.h>
#endif
#ifdef __LDBL_COMPAT
# include <bits/stdio-ldbl.h>
#endif

__END_DECLS

#endif /* <stdio.h> included.  */

#endif /* !_STDIO_H */
```