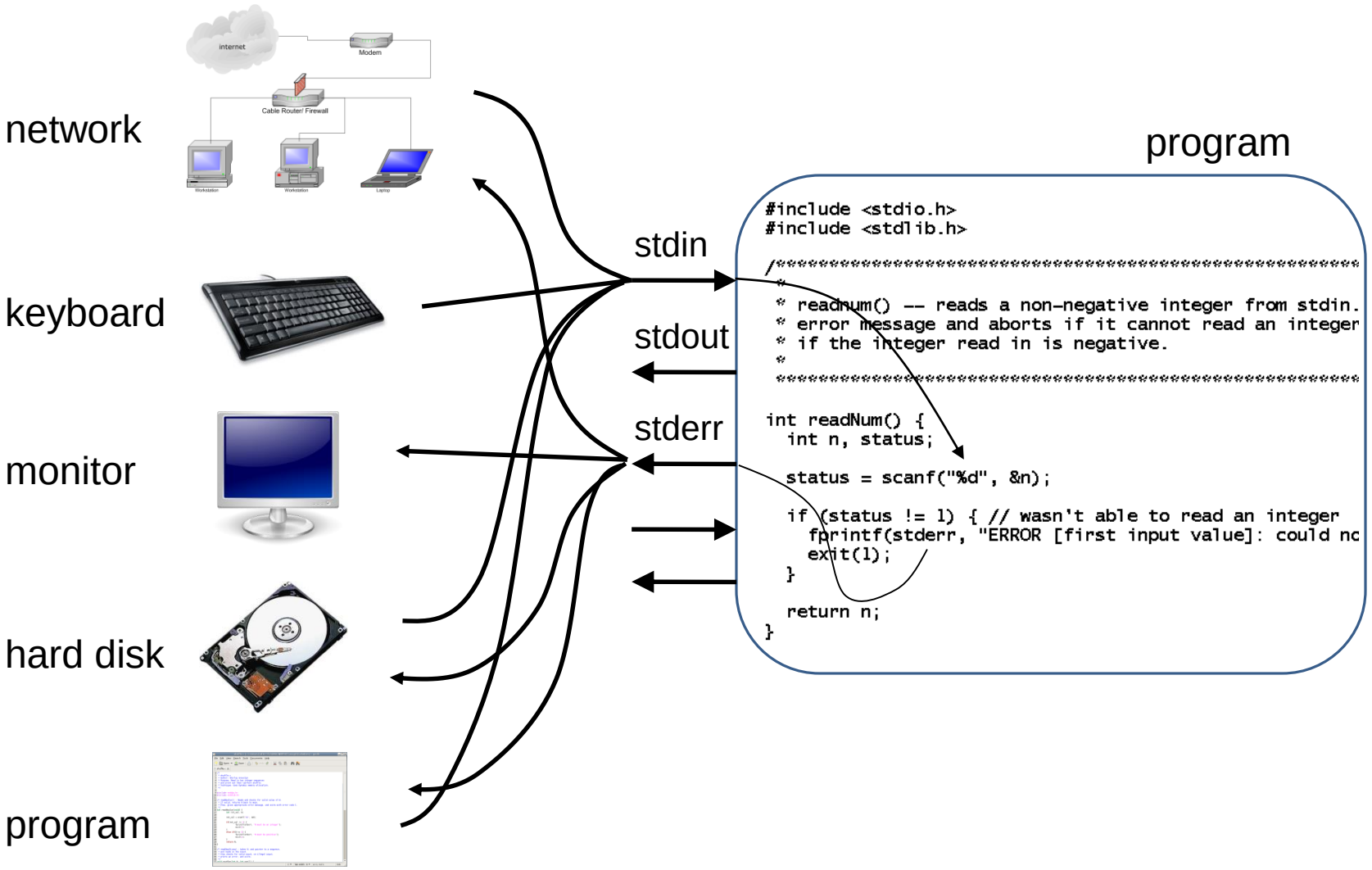
Streams

- A *stream* is any source of input or any destination for output
  - conceptually, just a sequence of bytes
  - accessed through a file pointer, which has type

    FILE *

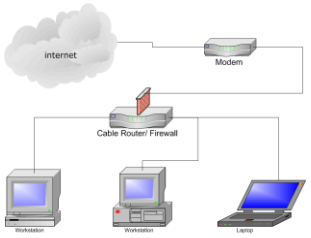
    however, not all streams are associated with files
  - three standard predefined streams: stdin, stdout, stderr

Streams


network
keyboard
monitor
hard disk
program
stdin
stdout
stderr
program
#include <stdio.h>
#include <stdlib.h>
/***
* readnum() -- reads a non-negative integer from stdin.
* error message and aborts if it cannot read an integer
* if the integer read in is negative.
****
int readNum() {
int n, status;
status = scanf("%d", &n);
if (status != 1) { // wasn't able to read an integer
fprintf(stderr, "ERROR [first input value]: could nc
exit(1);
}
return n;
}

Streams

network



keyboard

monitor

hard disk

program

stdin

stdout

stderr

program

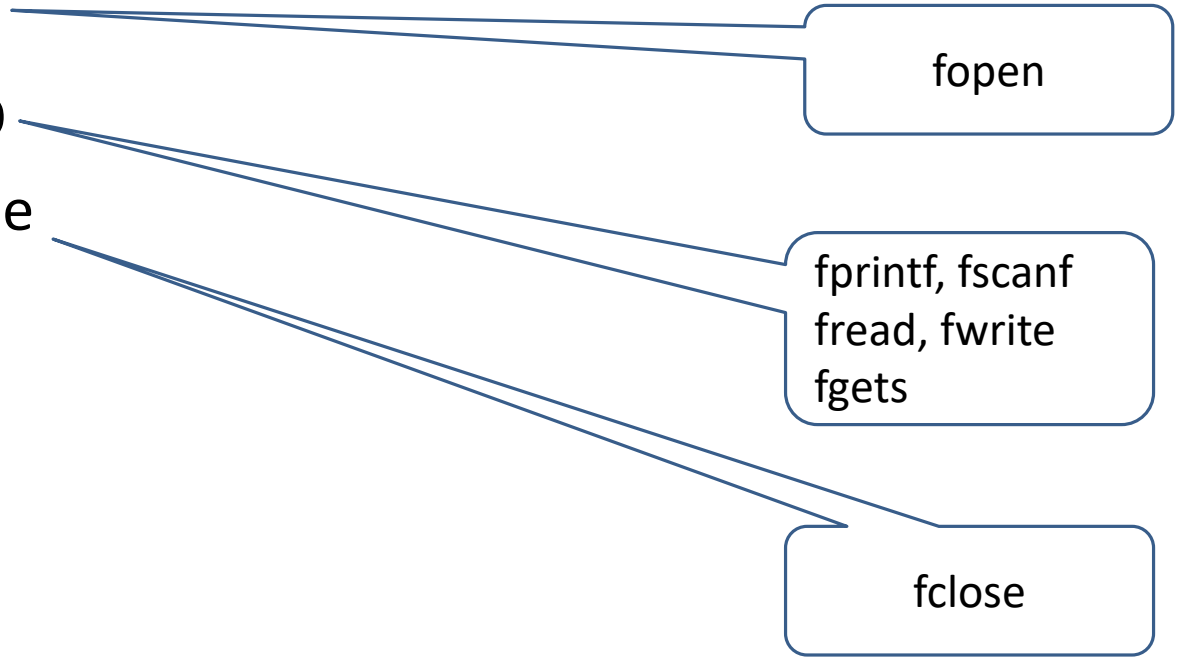
```
#include <stdio.h>
#include <stdlib.h>

/*****************************************************
 *
 * readnum() -- reads a non-negative integer from stdin.
 * error message and aborts if it cannot read an integer
 * if the integer read in is negative.
 *
 *****************************************************

int readNum() {
  int n, status;

  status = scanf("%d", &n);

  if (status != 1) { // wasn't able to read an integer
    fprintf(stderr, "ERROR [first input value]: could nc
    exit(1);
  }

  return n;
}
```

streams provide an abstraction of a byte sequence for I/O

Typical structure of I/O operations

A program's I/O operations usually have the following structure:

1. Open a file
2. Perform I/O
3. Close the file



Opening a file

FILE * fopen(char *filename, char *mode)

name of file to open

file pointer for the stream, if fopen succeeds;
NULL otherwise

“r”	read
“w”	write (file need not exist)
“a”	append (file need not exist)
“r+”	read and write, starting at the beginning
“w+”	read and write; truncate file if it exists
“a+”	read and write; append if file exists

Closing a file

```
int fclose(FILE *fp)
```

file pointer for
stream to be closed

return value:
0 if the file was closed successfully;
EOF otherwise

Code structure

```
FILE *fp;
...
fp = fopen(filename, "r");     // or whatever mode is appropriate
if (fp == NULL) {
   ... give error message and exit ...
}
... read and process file ...
status = fclose(fp);
if (status == EOF) {
   ... give error message...
}
```

Reading and writing

- **fprintf**, **fscanf**
  - similar to printf and scanf, with additional **FILE *** argument
- **fread**(*ptr*, *sz*, *num*, *fp*)
  - reads *num* elements, each of size *sz*, from stream *fp* and stores them at *ptr*
  - does not distinguish between end-of-file and error
    - use **feof**() and **ferror**()
- **fwrite**(*ptr*, *sz*, *num*, *fp*)
  - writes *num* elements, of size *sz*, from *ptr* into stream *fp*
- return values:
  - no. of items successfully read/written (not no. of bytes)

Example: fread

hed: /cs/www/classes/cs352/spring10/Code/ex.6.FileIO

```
% cat nlines.c
/*
 * File: nlines.c
 * Purpose: count the no. of lines in a given file; illustrate file I/O
 */
#include <stdio.h>

char buf[4096];
int nlines = 0;

int main(int argc, char *argv[]) {
  FILE *fp;
  int i, nchars;

  if (argc < 2) {
    fprintf(stderr, "Usage: %s file\n", argv[0]);
    return 1;
  }

  fp = fopen(argv[1], "r");
  if (fp == NULL) {
    fprintf(stderr, "Could not open file %s\n", argv[1]);
    return 1;
  }

  while (!feof(fp)) {
    nchars = fread(buf, sizeof(char), 4096/sizeof(char), fp);

    for (i = 0; i < nchars; i++) {
      if (buf[i] == '\n') {
        nlines += 1;
      }
    }
  }

  printf("File %s contains %d lines\n", argv[1], nlines);
  fclose(fp);
  return 0;
}
%
```

open the file for reading

check for EOF
(**fread** does not distinguish between EOF and error)

read from the file
(as much as will fit in the buffer)

close the file when done
(should check for errors)

Example: fscanf

hed: /cs/www/classes/cs352/spring10/Code/ex.6.FileIO

```
% cat add-numbers-in-file.c
/*
 * File: add-numbers-in-file.c
 * Purpose: add up the integers in a given file; illustrate file I/O
 */
#include <stdio.h>

int main(int argc, char *argv[]) {
  FILE *fp;
  int status, num, sum = 0;

  if (argc < 2) {
    fprintf(stderr, "Usage: %s file\n", argv[0]);
    return 1;
  }

  fp = fopen(argv[1], "r");
  if (fp == NULL) {
    fprintf(stderr, "Could not open file %s\n", argv[1]);
    return 1;
  }

  while ((status = fscanf(fp, "%d", &num)) != EOF) {
    if (status == 0) {
      fprintf(stderr, "Non-numeric data in input file %s\n", argv[1]);
      return 1;
    }

    sum += num;
  }

  printf("sum = %d\n", sum);
  fclose(fp);
  return 0;
}
%
```

fscanf

hed: /cs/www/classes/cs352/spring10/Code/ex.6.FileIO

```
% gcc -Wall add-numbers-in-file.c
%
% cat input-numbers
1
2
3
4
5
%
%
% ./a.out input-numbers
sum = 15
%
%
```

Errors

- fopen() can fail for many different reasons

hed: /cs/www/classes/cs352/spring10/Code/ex.5.FileIO

```
ERRORS
       EACCES The requested access to the file is not allowed, or search permission is denied for one of the
              directories in the path prefix of pathname, or the file did not exist yet and write access  to
              the parent directory is not allowed.  (See also path_resolution(7).)

       EEXIST pathname already exists and O_CREAT and O_EXCL were used.

       EFAULT pathname points outside your accessible address space.

       EFBIG  pathname  refers  to  a  regular  file,  too  large  to  be  opened;  see  O_LARGEFILE  above.
              (POSIX.1-2001 specifies the error EOVERFLOW for this case.)

       EISDIR pathname refers to a directory and the access requested involved writing (that is, O_WRONLY or
              O_RDWR is set).

       ELOOP  Too  many  symbolic  links were encountered in resolving pathname, or O_NOFOLLOW was specified
              but pathname was a symbolic link.

       EMFILE The process already has the maximum number of files open.

       ENAMETOOLONG
              pathname was too long.

       ENFILE The system limit on the total number of open files has been reached.

       ENODEV pathname refers to a device special file and no corresponding device exists.  (This is a Linux
              kernel bug; in this situation ENXIO must be returned.)

       ENOENT O_CREAT  is  not set and the named file does not exist.  Or, a directory component in pathname
              does not exist or is a dangling symbolic link.

       ENOMEM Insufficient kernel memory was available.

       ENOSPC pathname was to be created but the device containing pathname has no room for the new file.

       ENOTDIR
              A component used as a directory in pathname is not, in fact, a directory, or  O_DIRECTORY  was
              specified and pathname was not a directory.
:
```

Giving sensible error messages: perror

hed: /cs/www/classes/cs352/spring10/Code/ex.5.FileIO

```
PERROR(3)                    Linux Programmer's Manual                    PERROR(3)

NAME
       perror - print a system error message

SYNOPSIS
       #include <stdio.h>

       void perror(const char *s);

       #include <errno.h>

       const char *sys_errlist[];
       int sys_nerr;
       int errno;

   Feature Test Macro Requirements for glibc (see feature_test_macros(7)):

       sys_errlist, sys_nerr: _BSD_SOURCE

DESCRIPTION
       The  routine  perror()  produces  a  message  on the standard error output, describing the last error
       encountered during a call to a system or library function.  First (if s is not NULL and *s is  not  a
       null  byte  ('\0')) the argument string s is printed, followed by a colon and a blank.  Then the mes-
       sage and a new-line.

       To be of most use, the argument string should include the name of  the  function  that  incurred  the
       error.   The  error  number is taken from the external variable errno, which is set when errors occur
       but not cleared when non-erroneous calls are made.

       The global error list sys_errlist[] indexed by errno can be used to obtain the error message  without
       the  newline.   The  largest  message  number  provided in the table is sys_nerr -1.  Be careful when
       directly accessing this list because new error values may not have been added to sys_errlist[].

       When a system call fails, it usually returns -1 and sets the variable errno  to  a  value  describing
       what went wrong.  (These values can be found in <errno.h>.)  Many library functions do likewise.  The
       function perror() serves to translate this error code into human-readable form.  Note that  errno  is
       undefined  after  a  successful library call: this call may well change this variable, even though it
       succeeds, for example because it internally used some other library function that failed.  Thus, if a
:
```

perror

hed: /cs/www/classes/cs352/spring10/Code/ex.6.FileIO

```
 * File: nlines-perror.c
 * Purpose: count the no. of lines in a given file; illustrate file I/O
 * as well as the use of perror() to give sensible error messages.
 */
#include <stdio.h>
#include <errno.h>

char buf[4096];
int nlines = 0;
int errno;

int main(int argc, char *argv[]) {
  FILE *fp;
  int i, nchars;

  if (argc < 2) {
    fprintf(stderr, "Usage: %s file\n", argv[0]);
    return 1;
  }

  fp = fopen(argv[1], "r");
  if (fp == NULL) {
    perror(argv[1]);   //WAS: fprintf(stderr, "Could not open file %s\n", argv[1]);
    return 1;
  }

  while (!feof(fp)) {
    nchars = fread(buf, sizeof(char), 4096/sizeof(char), fp);
     for (i = 0; i < nchars; i++) {
      if (buf[i] == '\n') {
        nlines += 1;
      }
    }
  }

  printf("File %s contains %d lines\n", argv[1], nlines);
  fclose(fp);
  return 0;
}
%
```

perror

hed: /cs/www/classes/cs352/spring10/Code/ex.6.FileIO

```
% gcc -Wall nlines-perror.c
%
% ls -al
total 28
drwxr-xr-x  2 debray dept 4096 2010-02-22 09:52 ./
drwxr-xr-x 10 debray dept 4096 2010-02-19 16:04 ../
-rwxr-xr-x  1 debray dept 7879 2010-02-22 09:52 a.out*
--w-------  1 debray dept    0 2010-02-22 09:45 FILE-WITH-NO-READ-PERMISSIONS
-rw-r--r--  1 debray dept   35 2010-02-19 16:47 infile0
-rw-r--r--  1 debray dept  704 2010-02-19 16:50 nlines.c
-rw-r--r--  1 debray dept  830 2010-02-22 09:47 nlines-perror.c
%
% ./a.out ./NONEXISTENT-FILE
./NONEXISTENT-FILE: No such file or directory
%
% ./a.out FILE-WITH-NO-READ-PERMISSIONS
FILE-WITH-NO-READ-PERMISSIONS: Permission denied
%
%
```