Table 22.1
Standard Streams

File Pointer	*Stream*	*Default Meaning*
`stdin`	Standard input	Keyboard
`stdout`	Standard output	Screen
`stderr`	Standard error	Screen

The functions that we've used in previous chapters—`printf`, `scanf`, `putchar`, `getchar`, `puts`, and `gets`—obtain input from `stdin` and send output to `stdout`. By default, `stdin` represents the keyboard; `stdout` and `stderr` represent the screen. However, many operating systems allow these default meanings to be changed via a mechanism known as ***redirection***.

Q&A

Typically, we can force a program to obtain its input from a file instead of from the keyboard by putting the name of the file on the command line, preceded by the < character:

```
demo <in.dat
```

This technique, known as ***input redirection***, essentially makes the `stdin` stream represent a file (`in.dat`, in this case) instead of the keyboard. The beauty of redirection is that the `demo` program doesn't realize that it's reading from `in.dat`; as far as it knows, any data it obtains from `stdin` is being entered at the keyboard.

Output redirection is similar. Redirecting the `stdout` stream is usually done by putting a file name on the command line, preceded by the > character:

```
demo >out.dat
```

All data written to `stdout` will now go into the `out.dat` file instead of appearing on the screen. Incidentally, we can combine output redirection with input redirection:

```
demo <in.dat >out.dat
```

The < and > characters don't have to be adjacent to file names, and the order in which the redirected files are listed doesn't matter, so the following examples would work just as well:

```
demo < in.dat > out.dat
demo >out.dat <in.dat
```

One problem with output redirection is that *everything* written to `stdout` is put into a file. If the program goes off the rails and begins writing error messages, we won't see them until we look at the file. This is where `stderr` comes in. By writing error messages to `stderr` instead of `stdout`, we can guarantee that those messages will appear on the screen even when `stdout` has been redirected. (Operating systems often allow `stderr` itself to be redirected, though.)

Text Files versus Binary Files

`<stdio.h>` supports two kinds of files: text and binary. The bytes in a ***text file*** represent characters, making it possible for a human to examine the file or edit it.