Opening a File | Coursera

coursera.org/learn/interacting-system-managing-memory/supplement/Aj8DG/opening-a-file

Opening a File

Opening a File

The first thing we must do to access a file (whether for reading or writing) is *open* it. Opening a file results in a *stream* associated with that file. A stream (which is represented by a**FILE *** in C) is a sequence of data (in this case **char**s), which can be read and/or written. The stream has a current position which may typically advances whenever a read or write operation is performed on the stream (and may or may not be arbitrarily movable by the program, depending on what the stream is associated with).

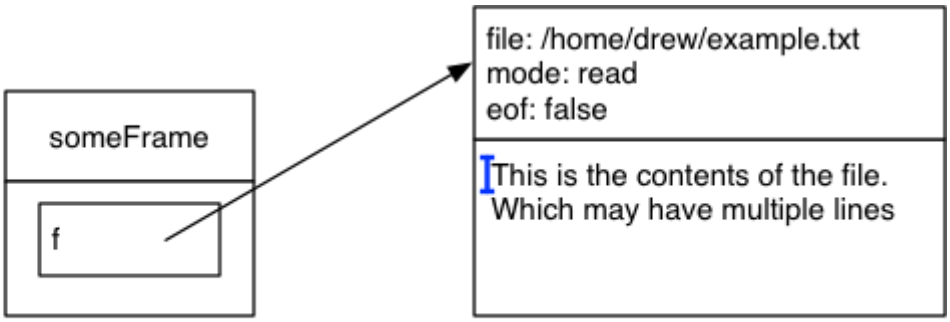
Typically, you will open a file with the **fopen** function, which has the following prototype:

```
1
FILE * fopen(const char * filename, const char * mode);
```

This function takes two arguments, the first of which is the name of the file to open. This filename is a string (which must be null-terminated, and is not modified by **fopen**), which is the pathname of the file to open. The pathname can either be an absolute path (starting with a / [forward slash]), or a path relative to the current working directory of the program. The second argument is the mode of the file—whether the file should be opened for reading and/or writing, whether to create the file if it does not exist, whether or not existing content is cleared out, and from what position accesses to the file start. Typically, the a string literal (such as "r") is passed for the **mode** parameter (though any expression which evaluates to a valid string is, of course, legal). We will discuss the mode in more detail momentarily.

Before going into more details about the mode, it is useful to see the effects of opening a file depicted. The figure below shows what happens when *fopen* is used to open a file, and the resulting **FILE** *is assigned to a variable called **f**. The depiction of the FILE is conceptual here, as the actual struct is much more complex, and does not directly contain the data from the file. However, we do not need to know what the actual FILEstruct contains, only how to operate on it. In fact, even if we do know what is in a FILEstruct on

one particular system, we should not attempt to use such information to directly access the fields inside of it, as the implementation details are system dependent, and thus our code would be non-portable.




```
FILE * f = open ("/home/drew/example.txt", "r");
```

Our conceptual representation of the FILE struct does, however, show the relevant pieces for typical use—we can use this abstraction to think about what a program will do, and thus how to execute it by hand. There is some information about the state of the file (which file is opened, whether it can be read or written, and whether or not the program has attempted to read past the end of the file) shown in the top portion. The bottom portion shows the data in the file, along with the position of the stream (indicated by a blue cursor). Read and write operations will occur at the current position (*i.e.*, this blue cursor), and will advance it.

We will note that real FILEstructs do not contain the name of the file, but rather a *file descriptor*—a numeric identifier returned to the program by the operating system when the program makes the system call to open the file (namely, the **open**system call). The C library functions which operate on the stream pass this descriptor to the relevant system calls which perform the underlying input/output operations for them.

It is possible for **fopen**to fail—for example, the file you requested may not exist, or you may not have the permissions required to access it. Whenever **fopen** fails, it returns NULL (and sets **errno** appropriately). You should always check the return value of **fopen**to see if it is NULL or a valid stream before you attempt to use the stream.

Mode	Read and/or write	Does not exist?	Truncate?	Position
r	read only	fails	no	beginning
r+	read/write	fails	no	beginning
w	write only	created	yes	beginning
w+	read/write	created	yes	beginning

Mode	Read and/or write	Does not exist?	Truncate?	Position
a	writing	created	no	end
a+	read/write	created	no	end

The possible modes for **fopen** are summarized in the table above. The first column shows the possible legal values for the mode string. The second column shows whether the opened file can be read, written, or both. The third column shows what happens if the requested file does not exist—for some modes, it is created (if possible), and for others, the call to **fopen**fails. The fourth column shows whether the contents of the file are truncated to zero-length if the file already exists—that is, whether or not the existing contents of the file are discarded. The final column shows where the accesses start—at the beginning or end of the contents of the file. We note that for the a modes (which stands for append), all writes will write their data to the end of the file, even if the program explicitly moves the current position elsewhere.

Completed