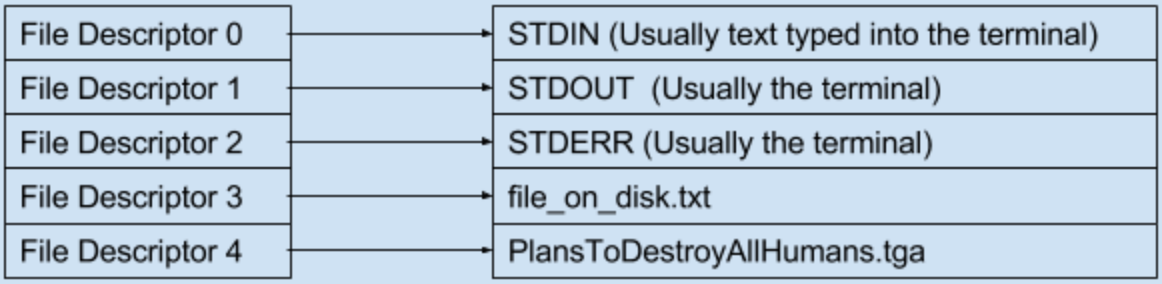
# Lab 12 Prelab: File I/O

One of the most important things in any programming language is the ability to read and alter files in the file system. While C does not really go out of its way to accommodate this sort of thing, UNIX and the POSIX standard do.

## Optional info: File Descriptors and the File Descriptor Table

Operating systems complying with the POSIX standards for how to build operating systems are required to give each process on the computer a table called the File Descriptor Table which stores a list of locations on the disk that files are located. If one opens a new file on a Linux computer, since Linux complies with POSIX, that means that a new entry in the File Descriptor Table will be made.

One interesting rule is that there are some special slots reserved in this File Descriptor Table. Slot 0 is called the standard input. Slot 1 is the standard output. Slot 2 is standard error output. When you start a program, the system will automatically open three files for the first three slots in the File Descriptor Table. If you are running your program from the terminal, by default these three files are actually the terminal.




...

**In C though, we are able to mostly ignore these File Descriptors as the function fopen() returns a FILE*, which actually manages the File Descriptors and your location in the file for us. We can then call fread() and it will put the entire file into a buffer of our choosing. REMEMBER: FILE* are not File Descriptors, and FILE* remember where you are in the file between reads. If you want to read from the start of the file again, you need to call rewind().**

# fopen()

To Open a File In C, one needs to call fopen like so:

```
FILE* myfile = fopen(“/path/to/file/goes/here”, “r+”);
```

This will make a file pointer to the file which  remembers our location in the file as well. The “r+” means we intend to both read and write this file. If we only want to read, we can omit the “+” in order to prevent accidental overwrites and speed things up.  If we want to read the whole file into memory right now, we can create a buffer via malloc and then call fread():

```
int* buff = malloc(sizeof(int) * 1000);
//assuming this file is 1000 ints long
fread(buff, sizeof(int), 1000, myfile);
//This will place 1000 things of size sizeof(int) into the
//buffer buff from the file.
```

Similarly, if we want to write our buffer back to our file, there is a function fwrite(). First though, we will need to call rewind() to set the file pointer to the start of the file again, since it has changed when we did our fread(), and is now 1000 ints into the file.

```
rewind(myfile);
fwrite(buff, sizeof(int), 1000, myfile);
```

If we want to close this file, we can call fclose(). This is very important if we have written something to the file, as C will often create its own internal buffer to store changes to files, and fclose() will make sure those buffers get written to disk accordingly.

```
fclose(myfile);
```

fread() is extreamly flexable and can read anything, even structs and strange data types. All you need to do is make sure you use sizeof() when setting the size of each element in your fread() call, and fread() will copy it into memory.