CS50 File I/O

Overview

Interacting with data stored on a computer's disk is integral to many programs. Data storage allows information to be used after the lifetime of a program without the hassle of inputting it again. Data is stored on a disk in the form of **files**, collections of data organized in such a way that the computer can understand. Files store a variety of media, including audio, text, movies, pictures, and more.

Key Terms

- file
- file pointer
- file extension
- I/O

Remember that all files are fundamentally just bits – 0s and 1s – that have been organized in a specific way. File types (a "video file," for instance) are merely abstractions of these bits. Likewise, **file extensions**, such as .txt, .c, or .mp3, merely act as references for computer programs that tell them what they should expect to find inside the file and what they should use the file for. So when a computer sees a certain extension, it knows that the corresponding file (i.e., its 0s and 1s) is formatted in a specific way and should be opened with an appropriate program.

Interacting With Files

The "**I/O**" in "file I/O" stands for **input/output**. Thus, file I/O refers to the process of retrieving information from and storing information in files. In general, the file I/O process, broadly, consists of a few steps. First, you must open the file, specifying the ways in which the file can be manipulated. After that, the file's data is free to be manipulated in any of the specified ways. Finally, the file must be closed!

Diving Into Code

Files are interacted with in C via "file pointers," i.e. references to files, which in code are the `FILE *` data type. New file references are usually initialized with the library function `fopen`, which takes two arguments (both strings): the filename and the mode for which the file should be opened.

There are many different ways to manipulate files. The most important of these are reading ("r"), writing ("w"), and appending ("a"), which is just like writing but done directly at the end of the file. To write to a file, we can use `fprintf`, specifying the file pointer to which we want to write and also what it is we want to write. Other functions, like `fgetc`, let us read from a file, getting characters, strings, and the like from the file pointer. Since there are many methods for writing and reading various types of data, we need to make sure to use the one which best suits our needs!

```
1  | // Write to file
2  | FILE *fp = fopen("document.txt", "w");
3  | fprintf(fp, "Hello, world!\n");
4  | fclose(fp);
5  |
6  | // Read from file
7  | fp = fopen("document.txt", "r");
8  | char c = fgetc(fp);
9  | while (c != EOF)
10 | {
11 |     printf("%c", c);
12 |     c = fgetc(fp);
13 | }
14 | fclose(fp);
```

Error Checking

It's also very important to understand and check for potential errors that might occur in the file I/O process. For instance, unlike the previous example, we should always make sure fopen was successful.

Take a look at the code at right and note the error checking to ensure `fopen` was indeed successful. Here, the mode specified was writing, or "w," so failures could occur if the file exists and it is corrupted. Had the mode been reading, or "r" (as in line 7 of the former code), errors could have resulted from trying to read from a nonexistent file. Also, not having the appropriate permission to open a file (to read from or to write to) will also return an error.

```
1 | // Write to file
2 | FILE *fp = fopen("document.txt", "w");
3 | // ensure the file was successfully opened
4 | if (fp == NULL)
5 | {
6 |     fprintf(stderr, "Error opening file.\n");
7 |     return 1;
8 | }
9 | // continue...
```