Reading a File with fread

coursera.org/learn/interacting-system-managing-memory/supplement/X09QK/reading-a-file-with-fread

We may also want to read non-textual data from a file. For example, we might have an image, video, sound, or other file where we write data in a binary format. In such a file, rather than writing the textual representation of an integer, the actual bytes for the integer are written to the file. The specific size of integer used for each piece of data is part of the file format specification. When we want to read data in this fashion, the most appropriate function is **fread**, which has the following prototype:

```
size_t fread (void * ptr, size_t size, size_t nitems, FILE * stream);
```

The first argument is a pointer to the data to write—it is a **void** *, since it could be any type of data. The next argument specifies the size of each item. That is, if we are writing an int or an array of ints, we would pass in **sizeof(int)**. However, if we are reading and writing files, we probably want to work with the inttypes defined in **stdint.h**, which have their sizes fixed across systems (such as **int32_t** or **uint64_t**). The third argument specifies how many such items should be read from the stream, and the final argument specifies from which stream to read. The **fread**function returns how many items were successfully read. As with **fgets**, you should employ **feof** and/or **ferror** to obtain more information if **fread** returns fewer items than you requested.

We will note that there is also a function, **fscanf**,which reads formatted input and performs conversions in the reverse fashion of what **printf** does. If you need this functionality, it is often easier to deal with errors if you use **fgets** (or, as we will see later, **getline**) and then use **sscanf** on the resulting string.**fgets** (and later **getline**) will read a line at a time, and then you can attempt to convert the results out, which may or may not have the desired format. In such a case, you can easily continue reading the next line. By contrast, **fscanf** will stop reading input as soon as it encounters something that does not match the requested format. If you want to continue reading from the next line, you must then explicitly read out the rest of the line before proceeding. If you want to know more about **fscanf** or **sscanf**, see their man pages.

✓

Completed