File I/O – fprint, fscanf, etc

If your application needs to perform any sort of I/O you must #include <stdio.h>. This provides a library of functions to access the keyboard, the screen, or the file system.

For instance, you can use standard output if you want to save a copy of your directory as a file.

`ls -l  > outputFile.txt`

will save your current directory to the file outputFile.txt

If you want to examine a file and copy it at the same time type:

`cat myFile | tee outFile.txt`

Catenates myFile to the screen and pipes it to the tee function, which saves the output to a disk file.

If your app requires input from the user, and you want to automate things, try:

`myApp < input.dat`

The next logical step would be to send the output to a file for later examination. Use:

`myApp < input.dat  > output.dat`

which takes input from your input.dat file, and sends the output to your output.dat file. You could write this as a long script, to test all of your applications automatically. Then run a test suite of input files against your application, while collecting all of the output, to check for errors. You could automate the error checks by using standard error to capture them.

I use printf() and scanf() most frequently. You have seen me use printf() to send variable's values to the screen, or to create an output report.

```
printf("x equals %5.3f \n", x);
```

will print out the floating point variable using the 5.3 format. If you don't want to move to the next line after printing, eliminate the \n character (newline).

```
Scanf(" 5.3f" &x);
```

will wait for floating point input from the user. A prompt to the user prior to this statement is always a good thing :)

The library stdio.h has printf() and scanf() duplicated in many forms. fprintf() and fscanf() are pretty much the same functions as before, but work on disk files, instead of from memory or the user. For these functions you need a file pointer defined, like this:

```
FILE *fp;
```

Now that you have a file pointer you need to pick which file to open, and which mode to use, like this:

```
fp = fopen("myData.dat", "r");
```

to read the file myData.dat or

```
fp = fopen("myData.dat", "a");
```

to append to myData if it exists, or create and write to it, if the file does not exist.

There are a few storage modes available, but since disk space is cheap, I normally read and write text files not binary ones. The binary mode is useful for security but I prefer legibility.

Make sure the file actually got opened by checking your file pointer fp.

```
if (fp == NULL)  // no file pointer returned
   {
   printf("Error!\n");
   exit(1);  // designate an error
   }
```

will exit if a file could not be opened.

Once myData.dat is open you can write to it with the function fprintf().

```
fprintf(fp, "5.3f", x);
```

```
fscanf(fp, "5.3f", &x);
```

will read the same information back again when you open the file for reading.

When you are done with your files you need to close them.

```
fclose(fp);
```

This will flush any buffers and complete the file I/O process. If you don't close your files before the end of the program you risk losing data.

I normally read an entire file into memory and manipulate it there. But, if you like, you can read portions of the file. In fact, you can move anywhere you want within the disk file, and read from there.

The commands ftell() and fseek() find the current position in the file or move to another position respectively. If you want to go back to the beginning of the file just use the command rewind(). If you're not sure of the size of your file simply read it character by character, until you reach the end of file. Check if you have reached the end of file (EOF) with the feof() command.

```
while ((c = fgetc(fp)) != EOF)   // standard C I/O file reading loop
   putchar(c);
```

Unwrap this statement from the inside. c = fgetc(fp) reads the file pointed at by fp and stores a single character into c. The comparison is made with != EOF which asks if c is equal to the end of file. If it is not the end of file, putchar(c) places c onto the output stream.

https://en.wikipedia.org/wiki/C_file_input/output

https://www.programiz.com/c-programming/c-file-input-output

https://www.tutorialspoint.com/cprogramming/c_file_io.htm

https://www.programiz.com/c-programming/c-file-input-output

https://en.wikipedia.org/wiki/C_file_input/output

https://www.tutorialspoint.com/cprogramming/c_file_io.htm

https://www.cprogramming.com/tutorial/cfileio.html

https://www.cs.bu.edu/teaching/c/file-io/intro/