File IO

UNIX/LINUX Files

UNIX (LINUX) handles files a little differently than other operating systems. In UNIX everything is a file and extensions (the thing after the period) don't really matter. In a shell (command line) you can open try to any file. The file might just look like a bunch of junk, but you can still open it.

You will see a lot of files that will have weird extensions like “.in”, “.out”, or even files with no extensions. Don't be afraid to try and open them up to see what is inside.

Standard Process

When working with files in C, there is a common procedure to follow. First you open the file, then you read/write from/to the file, and finally you close the file.

Opening

When you open a file, you provide the path to the file and this tells the computer to give you access to the file. The standard function to open a file is:

```
FILE* fopen(const char* filePath, const char* mode);
```

The 'const' on a parameter means that the function promises not to change the value inside of the function, so you don't have to worry too much about that.

filePath is the path to the file that you want to access. If the file is in the same directory as your program, then you can just use the files name. This can be a relative or absolute path.
Relative paths start from you current directory and you navigate to your file using folder names and '..' (up one folder).
Absolute paths start at the root of the files system and begin with a '/'.

The mode is what you want to do with the file. The most basic modes are “r” (read) and “w” (write).

Note that this returns a FILE*. You don't have to know what a FILE* is, you just have to know how to use it. Just keep it around as a variable.

Be careful, when you open a file for reading and the file exists than it gets deleted.

Ex:

```
FILE* myFileRead = fopen(“myFile.txt”, “r”);
  //open myFile.txt for reading
FILE* myFileWrite = fopen(“myFile.txt”, “w”);
  //open myFile.txt for writing
```

Reading/Writing

The next step is to actually read from the file. There are many ways to read from a file, I will cover two. Note are that unlike reading from the command line, you can go backwards in a file. However, you have to be careful and we won't do it in this class.

Reading

To see more ways to read from a file, run the command `man fgetc`.

fscanf

This works exactly like scanf, but you have to give it a FILE* as the first parameter.

```
fscanf(myFileRead, “%d %c”, &someInt, &someChar);
```

```
//Read an int and a char from myFile
```

fgetc

This will read a single char from a file and return it to you. You have to be a little careful, any char read will be returned, this includes spaces and newlines. This functions returns an int and not a char. If there are no more characters to read (your are at the end of the file) than fgetc will return EOF, otherwise it will return a char.

```
int c;
do
{
   c = fgetc(myFileRead);
   if (c != EOF)
   {
        //Do something
   }
} while (c != EOF);
```

Writing

To see more ways to write to a file, run the command `man fputc`.

fprintf

Just like printf, but print to a file. The first argument to this is the FILE*, everything else is just like printf.

```
fprintf(myFileWrite, "Stuff: %d\n", someInt);
```

fputc

Write a single char to a file.

```
fputc(myFileWrite, 'A'); //Write 'A' to myFile
```

Closing

The last thing that you need to do is close the file. If you don't close a file that you are writing to, than it may not actually get written.

```
fclose(myFileRead);
fclose(myFileWrite);
```

Program

I want you to write me a program that reads a file and the writes a new file with all occurrences of 'c' replaced with 'q'. You must have a header file for this program.

Specs

1. All input should be read from a file.
    1. You may #define the name of the file.
2. All output should be written to a file.
    1. You may also #define the name of this file.
3. Whenever you read a 'c' character, replace that with a 'q'.
    1. Besides the 'c's replaced with 'q's, the files should be identical.

Test Run

My input file:

```
Yo!
Its a file!
```

```
My name is Eric.
```

Output file:

```
Yo!
Its a file!
My name is Eriq.
```