Files

Two main types

1.) text files ← simpler & common usage
2.) binary files

What are they in C?

- files, and their contents, in C are considered and read as a sequential stream of bytes.

File Operations

- creating new files
- writing new files
- overwriting existing files
- reading from files
- opening/closing files

Steps (for working with files)

1.) create a pointer of file type
ex. `FILE * file_pointer;`

2.) "open" a file - associate it with our pointer - and define access (declare reading or writing permissions (or both))
syntax: `file_pointer = fopen(<file name>, <type of access/operation>);`
→ permissions: "w" - write
"r" - read
"a" - append
ex. `file_pointer = fopen("file_name.txt", "w");`

3.) ensure file opened successfully! If for any reason the fopen() function fails, it will return NULL, so it is important to check that this is not the case before working with the file.

4.) work with the file! However your code requires

5.) "close" the file (once done working with it). This is important for several reasons:
→ it eats up memory when not in use
→ avoids potential bugs with conflicting or overlapping streams (eg. reading and writing to a file simultaneously).

Files (Cont.)

Creating a File

Use the fopen() function

- if you give a path/file name which does not already exist, the file will be created.
- if the file already exists and the permission(s) passed into fopen() is write ("w"), it will overwrite the existing file.
- only works if given write ("w") permission(s).

Writing to a File

First:

- open the file with write ("w") permissions

Next:

- use any of the notable "put" functions listed below:
  - fputc(character, file pointer) – "puts" the given character into the file, at the end of the file.

    ex. `fputc('a', file_pointer);`

  - fprintf(file pointer, placeholders, parameters) – much like printf, takes something to print, but prints out into a given file/destination.

    ex.
    ```
    fprintf(file_pointer, "testing");
    fprintf(file_pointer, "\n\n%d", num);
    ```

  - fputs(string, file pointer) – takes a string and "puts" it into the given file.

    ex.
    ```
    fputs("hello", file_pointer);
    fputs(myStr, file_pointer);
    ```

Files (Cont. 2)

Reading a File

First:

- open the file with read ("r") permissions.

Next:

- use any of the notable "get" functions listed below:
  - fgetc(source) – "gets" the next character from the source
    ex. `char myChar = fgetc(file_pointer);`
  - fscanf(source, placeholders, parameters) – reads the formatted input from the file stream
    ex. `fscanf(file_pointer, "%c", &myChar);`
  - fgets(string, length limit, source) – reads a text line or string from the specified source; terminates when encounters a newLine.
    (string: variable to store result in)
    ex. `fgets(myStr, 10, file_pointer);`

EOF (End Of File) Indicator

- unlike the null character in strings ('\0'), the EOF character is not actually stored in the file itself.
- helpful for using loops, etc. to read through a file since it indicates when the end has been reached.
- feof() function – returns whether the given pointer position is the end of the file or not.

```
ex.  while (!feof(file_pointer)) {
         myChar = fgetc(file_pointer);
         printf("%c", myChar);
     }
     fclose(file_pointer);
```