File I/O

What is File I/O

File I/O intro

- File I/O is the process of reading and writing data to and from files. Files are used to **store** data permanently on a computer's hard drive.

- File I/O is an essential part of many different types of programs, including:
  - **Web browsers**: Web browsers use file I/O to download and display web pages and images.
  - **Word processors**: Word processors use file I/O to save and load documents.
  - **Databases**: Databases use file I/O to store and retrieve data.
  - **Video games**: Video games use file I/O to save and load game states, as well as to load textures and other game assets.

File Operations

fopen()

Opening Modes in Standard I/O

File Mode	Meaning of Mode	During Inexistence of file
r	Open for reading.	If the file does not exist, fopen() returns NULL.
w	Open for writing.	If the file exists, its contents are overwritten. If the file does not exist, it will be created.
a	Open for append. i.e, Data is added to end of file.	If the file does not exists, it will be created.
r+	Open for both reading and writing.	If the file does not exist, fopen() returns NULL.
w+	Open for both reading and writing.	If the file exists, its contents are overwritten. If the file does not exist, it will be created.
a+	Open for both reading and appending.	If the file does not exists, it will be created.

- To open a file in C, you use the `fopen()` function.
- The fopen() function takes two arguments:
  - the name of the file to open
  - the mode in which to open the file.

`fread()` and `fwrite()`

- Once you have opened a file, you can use the `fread()` function to **read data from the file** and the `fwrite()` function to **write data to the file**.
- The `fread()` function takes four arguments:
  - the pointer to the buffer where the data should be stored
  - the size of each element in the buffer
  - the number of elements to read
  - the file pointer.
- The `fwrite()` function takes four arguments:
  - the pointer to the data to be written
  - the size of each element in the data
  - the number of elements to write
  - the file pointer.

`fclose()`

- When you are finished reading or writing data to a file, you must close the file using the `fclose()` function.

File Pointers

- A file pointer is **a variable that stores the current position in a file**. When you read or write data to a file, the file pointer is updated to reflect the new position.
- File pointers are used in **all** file operations, including opening, closing, reading, and writing files.

Usage of file pointers

- When you **open** a file, the file pointer is set to the beginning of the file. This means that the next read operation will read the first byte in the file.
- When you **read** data from a file, the file pointer is updated to point to the next byte in the file. This means that the next read operation will read the next byte in the file, and so on.
- When you **write** data to a file, the file pointer is updated to point to the next byte in the file. This means that the next write operation will write the next byte in the file, and so on.
- You can also use the file pointer to move to a specific position in the file. For example, to move to the middle of the file, you can use the `fseek()` function.

- Remember, it's important to always check if the file pointer is NULL before using it, as NULL indicates that the file could not be opened.

Reading & Writing into files

1. `fscanf`: used to read formatted input from a file. It works similarly to `scanf`, but reads from a file instead of standard input.
2. `fprintf`: used to write formatted output to a file. It works similarly to `printf`, but writes to a file instead of standard output.
3. `fgetc`: used to read a single character from a file. It takes a file pointer as an argument and returns the character read.
4. `fputc`: used to write a single character to a file. It takes the character and a file pointer as arguments.
5. `fgets`: used to read a string from a file. It reads until either a newline character is encountered, the end of the file is reached, or the specified number of characters have been read, whichever comes first.
6. `fputs`: used to write a string to a file. It writes the string to the file up to but not including the null character.

These functions can be used to read and write data to any type of file, including text files, binary files, and image files.

Error Handling

In C, error handling during file operations is crucial to prevent crashes or unexpected behavior.

1. Checking if a file opened successfully

- When you open a file using `fopen`, it returns a file pointer. If the file cannot be opened for some reason (like the file doesn't exist or you don't have the necessary permissions), `fopen` returns `NULL`. Always check if the file pointer is `NULL` before proceeding with other file operations.

```
FILE *fp = fopen("file.txt", "r");
if (fp == NULL) {
    printf("Failed to open file.\n");
    // handle the error, e.g., exit the program
}
```

2. Checking for errors during read/write operations

- Functions like `fgetc`, `fputc`, `fgets`, `fputs`, `fread`, and `fwrite` return a special value (like `EOF` or `NULL`) when an error occurs.
- Always check the return value of these functions to ensure the operation was successful.

3. Use the `ferror()` function to check for errors

- The `ferror()` function returns a non-zero value if an error has occurred during a file operation.
- You can call the `ferror()` function after each file operation to check for errors.

```c
FILE *file = fopen("example.txt", "r");
if (file == NULL) {
    // Handle error
}

// Do some operations with the file...

if (ferror(file)) {
    // An error occurred while reading from or writing to the file
}
```

4. Use the `feof()` function to check for the end of the file

- The `feof()` function returns a non-zero value if the end of the file has been reached.
- You can use the `feof()` function to prevent reading or writing past the end of the file.
- In this example, feof(file) will return a non-zero value (which is considered as true in C) when the end of the file is reached. Otherwise, it will return 0 (false), and the loop will continue.

```c
FILE *file = fopen("file.txt", "r");
if (file == NULL)
{
    // handle error
}

// Read the file until the end
while (!feof(file))
{
    // process file
}

fclose(file);
```

5. Use the `clearerr()` function to clear error flags

- The `clearerr()` function clears the error flags for the specified file stream.
- You can call the `clearerr()` function after handling an error to clear the error flags and allow subsequent file operations to succeed.

```c
FILE *file = fopen("example.txt", "r");
if (file == NULL) {
    // handle error
}

// some operations on file that might cause an error or EOF

clearerr(file);
```

Resources

1. C Tutorial on File I/O
2. File I/O in C
3. Studytonight.com

See you at the next session!