File I/O in C

Md. Aminul Islam Shazid

06 Dec 2025

Outline

1. Introduction
2. Example
3. Summary

Introduction

File Input-Output (I/O)

Basic reasons we use files:

- Store results permanently
- Process large datasets that cannot be typed in manually
- Exchange data with other programs

Two broad file categories:

- Text files: store data as human-readable characters
- Binary files: store raw bytes (we will not cover details here)

FILE Pointer

- Files are accessed using file pointers: `FILE *`
- A `FILE *` variable does not hold actual file content
- It holds information about the connection: current position, status, buffering, mode, etc.

Life Cycle of using a File

1. Open the file using `fopen()`
2. Read from or write to the file
3. Close the file using `fclose()`

Failing to close a file may lead to:

- Data not being fully written (due to buffering)
- Running out of available file descriptors

Opening Files

To work with files, we must first open them using `fopen()`.

```
FILE *fp = fopen("filename.txt", "mode");
```

Basic modes:

- `"r"`: open for reading; fails if file does not exist
- `"w"`: open for writing; creates new file or overwrites existing file
- `"a"`: open for appending (adding new contents to existing file); creates file if needed

Opening Files (cont.)

Always check if opening succeeded:

```
if(fp == NULL){
    // error
}
```

Common reasons for failure include:

- File does not exist (when using `"r"`)
- Insufficient permissions
- Incorrect path/file name

Closing Files

Closing files is done using `fclose(fp);`.
Reasons to close files:

- Ensures buffered output is actually written to disk
- Frees system resources
- Prevents file corruption

If a program ends without closing files, the OS usually closes them, but:

- This is unreliable
- Buffered content may be lost
- Considered bad programming practice

Example

Writing to a File

Writing must come first so the next example can read the file.
Common functions:

- `fprintf()`: write formatted text.
- `fputs()`: write a string.
- `fputc()`: write a single character.

Example: Writing to a File

```c
#include <stdio.h>

int main(){
    FILE *fp = fopen("data.txt", "w");
    if(fp == NULL){
        printf("Cannot open file.\n");
        return 1;
    }
    fputs("Hellow, world!\n", fp);
    fprintf(fp, "This is an integer: %d\n", 10);
    fputc('X', fp);    // single character

    fclose(fp);
    return 0;
}
```

Reading from a File

Now we read the file created by the previous program.
Functions:

- `fscanf()`: formatted reading
- `fgets()`: read a line
- `fgetc()`: read a single character

Example: Reading from a File

```c
#include <stdio.h>

int main(){
    FILE *fp = fopen("data.txt", "r");
    if(fp == NULL){
        printf("Cannot open file.\n");
        return 1;
    }
    char buffer[100];
    while(fgets(buffer, sizeof(buffer), fp) != NULL){
        printf("%s", buffer);
    }
    fclose(fp);
    return 0;
}
```

Summary

Basic Error Handling

Important situations to detect:

- **Failed fopen**: always check if pointer returned is `NULL`
- **Reaching EOF**: functions return special values (e.g., `NULL` for `fgets`, `EOF` for `fgetc`)
- **Read errors**: typically uncommon in small programs but detectable using `ferror()`

EOF vs Error:

- EOF means the file ended normally
- Errors indicate issues such as hardware failure or file being removed during reading

In basic programs, it is usually enough to check for `NULL` or `EOF`

Summary

- Use `fopen()` to open files with the correct mode
- Perform reading or writing using `fprintf()`, `fputs()`, `fgetc()`, `fgets()`, or `fscanf()`
- Always close files using `fclose()`
- Always check if opening a file failed

These are the essential basics required before moving to advanced topics like binary files and random access.