## 🧾 ****Introduction****

File handling in C is a fundamental concept that allows a program to **store, retrieve, and manipulate data permanently** on the system's storage (like HDD or SSD), unlike variables that only exist during runtime. Whether it's saving logs, configuration files, student records, or binary data, file handling enables interaction with the outside world.

In C, **file operations** are made possible through the **Standard I/O Library (**stdio.h**)**, which provides a set of predefined functions and structures to work with files. File handling is crucial in real-world embedded systems, firmware applications, and system software where interaction with non-volatile memory, configuration logs, or report generation is needed.

Through file handling, you can:

* Read from a file (fopen, fscanf, fgets, etc.)
* Write to a file (fprintf, fputs, fwrite)
* Update existing content
* Navigate within a file (fseek, ftell, rewind)
* Handle errors gracefully using ferror() and feof()

## 📚 ****Types of Files in C****

There are two main types of files in C:

1. **Text Files**
   * Stored as readable characters (ASCII)
   * Editable by text editors (e.g., .txt, .log)
   * Uses functions like fscanf(), fprintf(), fgets(), etc.
2. **Binary Files**
   * Stored as raw binary data (compact and fast)
   * Not human-readable
   * Uses fread(), fwrite()

## 🧱 ****Core File Operations****

| Operation | Function |
| --- | --- |
| Create/Open | fopen() |
| Read | fscanf(), fgets(), fgetc() |
| Write | fprintf(), fputs(), fputc() |
| Close | fclose() |
| Error Handling | ferror(), feof() |
| Move Cursor | fseek(), ftell(), rewind() |

## 🗃️ FILE \* ****– File Pointer****

All file operations in C are done using a FILE pointer:

FILE \*fp;

This pointer stores information about the file (like current position, mode, etc.) and is returned by fopen().

## 📝 ****fopen() – Opening a File****

FILE \*fopen(const char \*filename, const char \*mode);

| Mode | Meaning |
| --- | --- |
| "r" | Open for reading (file must exist) |
| "w" | Open for writing (creates or truncates file) |
| "a" | Open for appending |
| "r+" | Open for reading and writing |
| "w+" | Open for read/write (truncates file) |
| "a+" | Open for reading and appending |

### ✅ Example:

FILE \*fp = fopen("data.txt", "w");

if (fp == NULL) {

printf("File couldn't be opened.\n");

return 1;

}

## ****Using Absolute Path (Full Directory Path)****

### 🖥️ Windows Example:

FILE \*fp = fopen("C:\\Users\\Ashwini\\Documents\\report.txt", "w");

🧠 Note:

* Use **double backslashes** \\ to escape Windows path separators.
* Single backslash \ is an **escape character** in C.

### 🐧 Linux/Unix Example:

FILE \*fp = fopen("/home/ashwini/Documents/report.txt", "w");

🧠 Note:

* Linux/macOS paths use regular **forward slashes** /
* Case-sensitive paths!

## 🖊️ ****Writing to Files****

### fprintf() – Like printf(), but for files:

fprintf(fp, "Name: %s\n", name);

### fputs() – Writes string to file:

fputs("Hello World!\n", fp);

### fputc() – Writes a single character:

fputc('A', fp);

## 📖 ****Reading from Files****

### fscanf() – Like scanf():

fscanf(fp, "%d %s", &age, name);

### fgets() – Reads a line:

fgets(buffer, 100, fp);

### fgetc() – Reads a character:

char ch = fgetc(fp);

## ❌ ****fclose() – Closing the File****

Always close the file to free resources:

fclose(fp);

## 🔁 ****File Navigation Functions****

| Function | Use |
| --- | --- |
| fseek(fp, offset, origin) | Moves file pointer |
| ftell(fp) | Tells current position |
| rewind(fp) | Goes to beginning of file |

### Example:

fseek(fp, 0, SEEK\_END); // Go to end of file

long size = ftell(fp); // Get file size

rewind(fp); // Go to beginning again

## ⚠️ ****Error Handling in File I/O****

### Check for EOF (end of file):

if (feof(fp)) {

printf("Reached end of file.\n");

}

### Check for error:

if (ferror(fp)) {

printf("Error reading the file.\n");

}

## 💡 ****Mini Example: Write and Read a File****

#include <stdio.h>

int main() {

FILE \*fp = fopen("hello.txt", "w");

if (fp == NULL) return 1;

fprintf(fp, "Hello Embedded World!");

fclose(fp);

fp = fopen("hello.txt", "r");

char ch;

while ((ch = fgetc(fp)) != EOF)

putchar(ch);

fclose(fp);

return 0;

}

## 🧠 ****Best Practices****

* Always check if the file opened successfully (fp == NULL)
* Don’t forget to fclose() to prevent memory leaks
* Use fgets() instead of fscanf() for safe string input
* Prefer "a" mode for logging to avoid overwriting

## 🏁 ****Conclusion****

File handling is a vital skill for any serious C programmer. Whether you're building an embedded logger, a data recorder, or a configuration manager, knowing how to read and write files gives your programs the power to **persist data beyond runtime**. In embedded systems, you may later replace standard file I/O with EEPROM, SD card, or Flash-based storage access using similar concepts.

Would you like this in a downloadable .txt, .md, or .pdf format?  
I can also generate **cheat sheets**, **practice exercises**, or a **project using file I/O**.