**🧠 Unit: File Handling in C**

This includes **definitions, syntax, easy examples, and outputs** — perfect for **BCA 1st Semester (FOP Subject – GTU)** students.

**🔹 1. Introduction to Files in C**

**Definition**

A **file** is a collection of data stored on a disk.  
In C, files are used to **store input/output data permanently**, even after the program ends.

**Why use files?**

* To store data permanently
* To read/write large amounts of data easily
* To share or reuse information later

**🔹 2. File Declaration**

Before using any file, you must declare a **file pointer** of type FILE.

**Syntax**

FILE \*file\_pointer;

**Example**

FILE \*fp;

The FILE type is defined in the **stdio.h** header file.

**🔹 3. Opening and Closing a File**

**Opening a File**

You must open a file before performing any operation (read/write).

**Syntax**

fp = fopen("filename", "mode");

**Modes in C File Handling**

| **Mode** | **Meaning** | **Action** |
| --- | --- | --- |
| "r" | Read | Opens file for reading (must exist) |
| "w" | Write | Creates a new file or overwrites existing |
| "a" | Append | Adds data to end of file |
| "r+" | Read + Write | Opens file for both (must exist) |
| "w+" | Write + Read | Creates new, allows both |
| "a+" | Append + Read | Opens existing or creates new |

**Closing a File**

Always close the file after use.

fclose(fp);

**🔹 4. Working with Text Files**

Text files contain readable characters (like .txt files).

**Example 1: Writing to a Text File**

#include <stdio.h>

int main() {

FILE \*fp;

fp = fopen("student.txt", "w");

if (fp == NULL) {

printf("Error opening file!");

return 1;

}

fprintf(fp, "Name: Meera\n");

fprintf(fp, "Marks: 85\n");

fclose(fp);

printf("Data written successfully!");

return 0;

}

**Output**

Data written successfully!

🧩 **File Content (student.txt):**

Name: Meera

Marks: 85

**Example 2: Reading from a Text File**

#include <stdio.h>

int main() {

FILE \*fp;

char ch;

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

if (fp == NULL) {

printf("File not found!");

return 1;

}

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

printf("%c", ch);

}

fclose(fp);

return 0;

}

**Output**

Name: Meera

Marks: 85

**🔹 5. Working with Binary Files**

Binary files store data in **binary (non-readable)** form — faster and more compact than text files.

**Example: Write and Read Binary File**

#include <stdio.h>

struct student {

char name[20];

int marks;

};

int main() {

struct student s1 = {"Ravi", 90}, s2;

FILE \*fp;

fp = fopen("student.dat", "wb");

fwrite(&s1, sizeof(s1), 1, fp);

fclose(fp);

fp = fopen("student.dat", "rb");

fread(&s2, sizeof(s2), 1, fp);

fclose(fp);

printf("Name: %s\nMarks: %d", s2.name, s2.marks);

return 0;

}

**Output**

Name: Ravi

Marks: 90

🧩 **Note:** Binary files are not readable in Notepad — data is stored in binary format.

**🔹 6. Character Input and Output**

Used to handle files **character by character**.

| **Function** | **Description** | **Example** |
| --- | --- | --- |
| fgetc(fp) | Reads a character | ch = fgetc(fp); |
| fputc(ch, fp) | Writes a character | fputc('A', fp); |

**Example: Character by Character Write & Read**

#include <stdio.h>

int main() {

FILE \*fp;

char ch;

fp = fopen("sample.txt", "w");

printf("Enter text (type @ to stop):\n");

while ((ch = getchar()) != '@') {

fputc(ch, fp);

}

fclose(fp);

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

printf("\nFile Content:\n");

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

putchar(ch);

}

fclose(fp);

return 0;

}

**Output**

Enter text (type @ to stop):

Hello Students@

File Content:

Hello Students

**🔹 7. End of File (EOF)**

**Definition**

EOF is a special value that indicates **no more data** is available to read from a file.

**Example**

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

printf("%c", ch);

}

**🔹 8. feof() Function**

Used to check if the end of the file has been reached.

**Syntax**

feof(file\_pointer);

**Example**

#include <stdio.h>

int main() {

FILE \*fp;

char ch;

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

while (!feof(fp)) {

ch = fgetc(fp);

printf("%c", ch);

}

fclose(fp);

return 0;

}

**🔹 9. Files of Records**

We can store **multiple structures (records)** in a file.  
Useful for applications like student databases, employee systems, etc.

**Example: Writing and Reading Records**

#include <stdio.h>

struct student {

int id;

char name[20];

float marks;

};

int main() {

struct student s;

FILE \*fp;

fp = fopen("record.txt", "w");

for (int i = 0; i < 2; i++) {

printf("Enter ID, Name and Marks: ");

scanf("%d %s %f", &s.id, s.name, &s.marks);

fprintf(fp, "%d %s %.2f\n", s.id, s.name, s.marks);

}

fclose(fp);

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

printf("\n--- Student Records ---\n");

while (fscanf(fp, "%d %s %f", &s.id, s.name, &s.marks) != EOF) {

printf("%d\t%s\t%.2f\n", s.id, s.name, s.marks);

}

fclose(fp);

return 0;

}

**Output**

Enter ID, Name and Marks: 1 Meera 88

Enter ID, Name and Marks: 2 Ravi 92

--- Student Records ---

1 Meera 88.00

2 Ravi 92.00

**🔹 10. Random Access to Files of Records**

**Definition**

Random access means directly reading or writing a **specific record** without reading the whole file.

This is done using:

* fseek() – moves the file pointer to a specific position
* ftell() – returns the current position of the file pointer
* rewind() – moves the file pointer to the beginning

**Syntax**

fseek(fp, position, origin);

ftell(fp);

rewind(fp);

| **Origin** | **Description** |
| --- | --- |
| SEEK\_SET | Beginning of file |
| SEEK\_CUR | Current position |
| SEEK\_END | End of file |

**Example: Random Access**

#include <stdio.h>

struct student {

int id;

char name[20];

float marks;

};

int main() {

struct student s;

FILE \*fp;

fp = fopen("students.dat", "rb");

fseek(fp, sizeof(struct student), SEEK\_SET); // move to 2nd record

fread(&s, sizeof(s), 1, fp);

printf("2nd Student Record:\n");

printf("ID: %d\nName: %s\nMarks: %.2f\n", s.id, s.name, s.marks);

fclose(fp);

return 0;

}

**✅ Summary Table**

| **Concept** | **Function** | **Purpose** |
| --- | --- | --- |
| fopen() | Opens a file | "r", "w", "a" |
| fclose() | Closes a file | — |
| fgetc() | Reads a character | — |
| fputc() | Writes a character | — |
| fprintf() | Writes formatted data | — |
| fscanf() | Reads formatted data | — |
| feof() | Checks end of file | — |
| fseek() | Moves file pointer | Random access |
| fwrite(), fread() | Binary read/write | — |