PROGRAMMING IN C

Chapter Notes

Chapter: User Defined Files

1. Introduction to Files

All the data that a program stores in ordinary variables is held in RAM, and is lost the moment the program terminates. A file provides a way of storing data permanently on a secondary storage device (such as a disk), so that it can be reused, updated, or shared even after the program that created it has finished running.

In C, every file operation is carried out through a special pointer called a file pointer, declared using the FILE structure defined in the header file stdio.h. Working with a file in C generally involves three basic steps, in order: opening the file, processing it (reading from it or writing to it), and finally closing it.

Files handled in C are broadly of two kinds:

- Text files — store data as a sequence of human-readable ASCII characters, organised into lines separated by a newline character.
- Binary files — store data exactly as it is represented in memory, as a sequence of raw bytes, and are not directly readable as plain text.

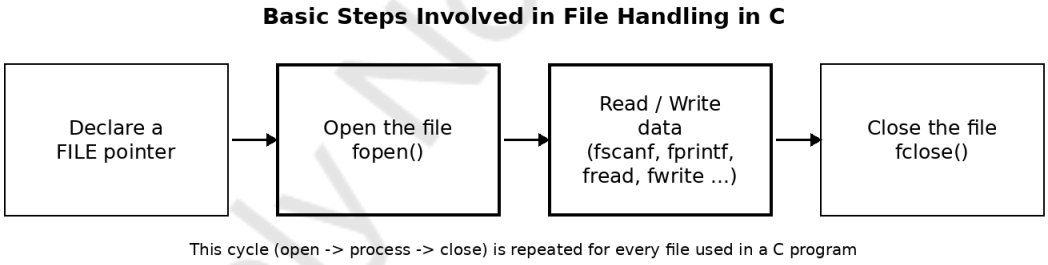

Fig. 1 — Basic steps involved in file handling in C

2. Different Modes for Opening Files

A file is opened in C using the fopen() function, which takes the file name and a mode string, and returns a file pointer that is used for all further operations on that file:

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

The mode string tells the compiler how the file is intended to be used — for reading, writing, appending, or a combination of these. The commonly used modes are summarised below:

Mode	Meaning
r	Opens an existing text file for reading only. Fails (returns NULL) if the file does not exist.
w	Opens a text file for writing only. Creates a new file if it does not exist, or truncates (erases) an existing one.
a	Opens a text file for appending. Creates the file if it does not exist; writing always occurs at the end of the file.
r+	Opens an existing text file for both reading and writing. The file must already exist.
w+	Opens a text file for both reading and writing. Creates a new file or truncates an existing one.
a+	Opens a text file for both reading and appending. Creates the file if it does not exist.
rb, wb, ab	Same as r, w, a respectively, but the file is opened in binary mode instead of text mode.
rb+, wb+, ab+	Same as r+, w+, a+ respectively, but the file is opened in binary mode instead of text mode.

If fopen() is unable to open the file for the requested mode — for example, trying to read a file that does not exist — it returns NULL. It is good practice to always check the returned pointer before using it, so as to avoid working with an invalid file.

3. Using Formatted and Unformatted Files in C

Depending on how data is transferred to and from a file, file input/output in C can be classified as formatted or unformatted:

- Formatted (text) I/O — performed using functions such as fprintf() and fscanf(), which convert data between its internal (binary) form and a human-readable text form, in the same way printf() and scanf() do for the console.
- Unformatted (binary) I/O — performed using functions such as fread() and fwrite(), which copy data exactly as it is stored in memory, without any conversion to text.

The key differences between the two approaches are summarised below:

Basis of Comparison	Formatted (Text) Files	Unformatted (Binary) Files
Functions used	fprintf(), fscanf(), fputs(), fgets()	fwrite(), fread()
Data representation	Data is converted to and stored as human-readable ASCII characters	Data is stored exactly as it exists in memory (raw bytes)
Readability	Can be opened and read directly in any text editor	Not human-readable; appears as garbled characters in a text editor

Basis of Comparison	Formatted (Text) Files	Unformatted (Binary) Files
Speed	Slower, since every value must be converted to and from text form	Faster, since no text conversion is required
Storage space	Usually takes more space (e.g., the number 1234 takes 4 bytes as text)	Usually more compact (e.g., an int always takes exactly sizeof(int) bytes)
Precision	Floating-point values may lose precision during text conversion	Exact value is preserved, since the memory image is copied as it is
Portability	More portable across different machines and platforms	Less portable, since data layout can differ across systems

4. Reading Data from Files

C provides several functions for reading data from a file, each suited to a different kind of data:

- fscanf(fp, "format", &variables) — reads formatted data from a text file, in the same way scanf() reads from the keyboard.
- fgetc(fp) — reads a single character from a file at a time.
- fgets(str, n, fp) — reads a line of text (up to n-1 characters, or until a newline) into a string.
- fread(ptr, size, n, fp) — reads n items, each of size bytes, directly from a binary file into memory.

Example — reading formatted data with fscanf():

```
FILE *fp = fopen("marks.txt", "r");
int roll;
float marks;
while (fscanf(fp, "%d %f", &roll, &marks) != EOF) {
    printf("%d\t%.2f\n", roll, marks);
}
fclose(fp);
```

Example — reading a block of records with fread():

```
struct Student s;
FILE *fp = fopen("students.dat", "rb");
fread(&s, sizeof(s), 1, fp);
fclose(fp);
```

5. Writing Data to Files

Correspondingly, C provides matching functions for writing data to a file:

- fprintf(fp, "format", variables) — writes formatted data to a text file, in the same way printf() writes to the screen.
- fputc(ch, fp) — writes a single character to a file.
- fputs(str, fp) — writes a string (without a trailing newline being added automatically) to a file.
- fwrite(ptr, size, n, fp) — writes n items, each of size bytes, directly from memory into a binary file.

Example — writing formatted data with fprintf():

```
FILE *fp = fopen("marks.txt", "w");
fprintf(fp, "%d %.2f\n", 101, 78.5);
fclose(fp);
```

Example — writing a record with fwrite():

```
struct Student s = {101, "Ravi", 78.5};
FILE *fp = fopen("students.dat", "wb");
fwrite(&s, sizeof(s), 1, fp);
fclose(fp);
```

6. Functions for Random Access of Records

By default, files in C are accessed sequentially — that is, one record after another from the beginning. However, C also allows random access, where any record can be reached directly, without reading all the records before it. This is achieved mainly through the following functions:

- fseek(fp, offset, whence) — moves the file pointer to a specific byte position, given by offset, measured from a reference point whence, which may be SEEK_SET (beginning of file), SEEK_CUR (current position), or SEEK_END (end of file).
- ftell(fp) — returns the current position of the file pointer, as the number of bytes from the beginning of the file.
- rewind(fp) — resets the file pointer back to the very beginning of the file; it is equivalent to calling fseek(fp, 0, SEEK_SET).

For a file made up of fixed-size records (such as an array of structures written using fwrite()), the nth record can be reached directly as follows:

```
fseek(fp, (n - 1) * sizeof(struct Student), SEEK_SET);
fread(&s, sizeof(struct Student), 1, fp);
```

This calculates the exact byte offset of the required record and moves the file pointer there directly, making record access far more efficient than reading through the file sequentially.

Multiple Choice Questions (MCQs)

Answer the following questions by choosing the most appropriate option. The answer key is provided at the end of this section.

1. Which header file must be included in a C program to use file-handling functions?

A. <file.h>
B. <stdlib.h>
C. <stdio.h>
D. <string.h>

2. Which data type is used to declare a file pointer in C?

A. int
B. FILE
C. FILE *
D. struct file

3. Which function is used to open a file in C?

A. open()
B. fopen()
C. newfile()
D. openfile()

4. If fopen() fails to open a file, what value does it return?

A. 0
B. -1
C. NULL
D. EOF

5. Which mode opens a text file for writing, creating a new file or erasing the contents of an existing one?

A. "r"
B. "a"
C. "w"
D. "r+"

6. Which mode should be used to add new data at the end of an existing file without erasing its content?

A. "w"
B. "a"
C. "r"
D. "r+"

7. To open an existing file for both reading and writing, without erasing its content, which mode is used?

A. "w+"
B. "a+"
C. "r+"
D. "r"

8. Which suffix is added to a mode string to open a file in binary mode?

A. x
B. n

C. b

D. t

9. Which function is used to close an opened file in C?

A. close()

B. fclose()

C. closefile()

D. endfile()

10. Which function is used for formatted reading of data from a file?

A. fread()

B. fgets()

C. fscanf()

D. fgetc()

11. Which function is used for formatted writing of data to a file?

A. fwrite()

B. fprintf()

C. fputs()

D. fputc()

12. Which pair of functions is used for unformatted (binary) reading and writing of data blocks?

A. fscanf() and fprintf()

B. fgets() and fputs()

C. fread() and fwrite()

D. fgetc() and fputc()

13. Which function moves the file pointer to a specific position within a file, enabling random access?

A. ftell()

B. fseek()

C. rewind()

D. fflush()

14. Which function returns the current position of the file pointer within a file?

A. fseek()

B. ftell()

C. rewind()

D. fgetpos() only

15. Which function resets the file pointer back to the beginning of the file?

A. fseek()

B. ftell()

C. rewind()

D. reset()

Answer Key

Q. No.	Answer	Q. No.	Answer	Q. No.	Answer
1	**C**	6	**B**	11	**B**
2	**C**	7	**C**	12	**C**
3	**B**	8	**C**	13	**B**
4	**C**	9	**B**	14	**B**
5	**C**	10	**C**	15	**C**

Broad / Descriptive Questions

Answer the following questions in detail, with suitable examples and diagrams wherever applicable.

1. What is the need for file handling in C? Explain the basic steps involved in working with a file, with the help of a diagram.
2. Explain the different modes available for opening a file in C, along with the purpose of each mode.
3. Distinguish between formatted (text) files and unformatted (binary) files in C, giving suitable examples of the functions used with each.
4. Explain, with syntax and an example, how data is read from a file using fscanf() and fgets().
5. Explain, with syntax and an example, how data is written to a file using fprintf() and fputs().
6. What is the difference between fread()/fwrite() and fscanf()/fprintf()? When would you prefer one approach over the other?
7. What is meant by random access of a file? Explain the role of fseek(), ftell() and rewind() in achieving random access, with examples.
8. Write a short C program to open a file, write a few lines of text into it, and then close the file.
9. Write a short C program that opens an existing text file and reads its contents character by character until the end of file is reached.
10. Explain how the nth record can be directly accessed in a file of fixed-size records, using fseek() together with fread().