1. Introduction to File Handling

File handling in C language refers to the process of **creating, opening, reading, writing, and closing files** using C programs.
Files are used to **store data permanently** on secondary storage devices such as hard disks.

Unlike variables, file data is **not lost when the program ends**.

Definition

File handling is a mechanism that allows programs to store and retrieve data from files.

---

2. Need for File Handling

File handling is required because:

- Variables store data temporarily
- Large data cannot be stored in memory
- Data needs to be reused later
- Permanent storage is required
- Data sharing between programs is needed

---

3. Types of Files in C

C language supports **two types of files**:

1. **Text Files**
2. **Binary Files**

3.1 Text Files

- Store data in **character format**
- Human readable
- Each line ends with newline character

Examples:

- .txt
- .c

3.2 Binary Files

- Store data in **binary format**
- Not human readable
- Faster and more secure

Examples:

- .dat
- .bin

---

4. File Pointer

To work with files, C uses a **file pointer**.

Definition

A file pointer is a pointer of type FILE used to identify a file.

Declaration

```
FILE *fp;
```

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

---

5. Opening a File

The fopen() function is used to **open a file**.

Syntax

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

Example

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

---

6. File Opening Modes

Mode	Description
r	Read
w	Write
a	Append
r+	Read + Write
w+	Write + Read
a+	Append + Read

---

7. Closing a File

After file operations, the file must be closed.

Syntax

```
fclose(fp);
```

Closing a file:

- Saves data
- Frees memory
- Prevents data loss

---

8. Writing Data to a File

8.1 Using fprintf()

Writes formatted output to a file.

```
fprintf(fp, "Hello World");
```

8.2 Using fputc()

Writes a single character.

```
fputc('A', fp);
```

8.3 Using fputs()

Writes a string.

```
fputs("C Programming", fp);
```

---

9. Reading Data from a File

9.1 Using fscanf()

Reads formatted data.

```
fscanf(fp, "%s", str);
```

9.2 Using fgetc()

Reads a single character.

```
ch = fgetc(fp);
```

9.3 Using fgets()

Reads a string.

```
fgets(str, 50, fp);
```

---

10. Checking End of File (EOF)

The feof() function checks **end of file**.

Example

```
while(!feof(fp))
{
   ch = fgetc(fp);
   printf("%c", ch);
}
```

---

11. File Handling Program Example (Text File)

```
FILE *fp;
char str[50];

fp = fopen("test.txt", "w");
fputs("Welcome to C", fp);
fclose(fp);

fp = fopen("test.txt", "r");
fgets(str, 50, fp);
printf("%s", str);
fclose(fp);
```

---

12. Binary File Handling

Binary files store data in **raw binary format**.

12.1 Writing to Binary File (fwrite)

```
fwrite(&s, sizeof(s), 1, fp);
```

12.2 Reading from Binary File (fread)

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

---

13. Random Access in Files

Random access allows moving file pointer to any location.

13.1 fseek()

```
fseek(fp, offset, position);
```

Positions:

- SEEK_SET
- SEEK_CUR
- SEEK_END

13.2 ftell()

Returns current file pointer position.

```
pos = ftell(fp);
```

13.3 rewind()

```
Moves file pointer to beginning.
```

```
rewind(fp);
```

---

14. File Handling with Structures

Structures can be stored in files.

Example

```
struct Student
{
   int roll;
   char name[20];
};
```

Write:

```
fwrite(&s, sizeof(s), 1, fp);
```

Read:

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

---

15. Error Handling in Files

15.1 Checking File Open Error

```
if(fp == NULL)
{
   printf("File not found");
   exit(1);
}
```

15.2 perror() Function

```
perror("Error");
```

---

16. Advantages of File Handling

- Permanent data storage
- Handles large data
- Data sharing
- Backup and recovery
- Efficient memory usage

---

17. Limitations of File Handling

- Slower than memory operations
- Requires proper error handling
- File corruption risk
- Platform dependency (binary files)

---

18. Common Errors in File Handling

1. Forgetting to close file
2. Using wrong file mode
3. Reading from write-only file
4. Not checking NULL pointer
5. Incorrect use of EOF

---

19. Applications of File Handling

- Database systems
- Banking software
- Student record management

- Payroll systems
- File management utilities

---

20. Difference Between Text and Binary Files

Feature	Text File	Binary File
Readability	Human readable	Not readable
Speed	Slow	Fast
Size	Larger	Smaller
Data	Characters	Binary

---

21. Best Practices

- Always close files
- Check file open success
- Use binary files for structures
- Use text files for readable data
- Handle errors properly

---

22. Interview / Exam Important Points

- FILE is a structure
- File pointer is required
- fopen() opens a file
- fclose() closes a file
- Binary files use fread() & fwrite()

---

23. Conclusion

File handling is a crucial concept in C programming that allows permanent storage and retrieval of data. Understanding file types, file modes, and file functions is essential for developing real-world applications like databases, record management systems, and system software.