perplexity

Files- Different types of files in C, Opening & Closing a file, Writing to and

Reading from a file, Processing files, Library functions related to file
– fseek(), ftell(), fread(), fwrite().

teach me all this with appropriate points, differences to note, syntax and an example

Here's a comprehensive guide to **file handling in C**, including types of files, basic operations, and key library functions:

Types of Files in C

Type	Description	Extension	Human-readable?
Text Files	Store data as ASCII characters. Easy to read and edit.	`.txt`	Yes
Binary Files	Store data in binary (0s and 1s). Efficient and secure, not human-readable.	`.bin`	No

- **Text files** are simple and accessible but less secure and use more space.
- **Binary files** are compact, secure, and faster for large or complex data, but require programs to interpret their contents[1] [2].

Opening and Closing a File

Opening a File

- Use `fopen()`:

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

- **Modes**: `"r"` (read), `"w"` (write), `"a"` (append), `"rb"` (read binary), `"wb"` (write binary), etc[3].

Closing a File

- Use `fclose()`:

```
fclose(fp);
```

- Always close files to save changes and free resources[3].

Writing to a File

Text File Example

```
FILE *fptr = fopen("file.txt", "w");
if (fptr != NULL) {
    fprintf(fptr, "Hello, World!\n");
    fclose(fptr);
}
```

- `fprintf()` writes formatted text to a file.
- `"w"` mode overwrites or creates the file[4].

Appending

```
fptr = fopen("file.txt", "a");
fprintf(fptr, "More text\n");
fclose(fptr);
```

- `"a"` mode adds content to the end without deleting existing data[4].

Binary File Example

```
FILE *fptr = fopen("data.bin", "wb");
int arr[] = {1, 2, 3};
fwrite(arr, sizeof(int), 3, fptr);
fclose(fptr);
```

- `fwrite()` writes binary data from memory to file[5] [6].

Reading from a File

Text File Example

```
FILE *fptr = fopen("file.txt", "r");
char buffer[^100];
while (fgets(buffer, 100, fptr)) {
    printf("%s", buffer);
}
fclose(fptr);
```

- `fgets()` reads a line from the file into a buffer[7].

Binary File Example

```
FILE *fptr = fopen("data.bin", "rb");
int arr[^3];
fread(arr, sizeof(int), 3, fptr);
fclose(fptr);
```

- `fread()` reads binary data from file to memory[5] [6].

Processing Files: Key Library Functions

Function	Purpose	Syntax Example
`fopen()`	Open a file	`FILE *fp = fopen("file.txt", "r");`
`fclose()`	Close a file	`fclose(fp);`
`fprintf()`	Write formatted text	`fprintf(fp, "Hello");`
`fscanf()`	Read formatted text	`fscanf(fp, "%d", &num);`
`fwrite()`	Write binary data	`fwrite(arr, sizeof(int), n, fp);`
`fread()`	Read binary data	`fread(arr, sizeof(int), n, fp);`
`fseek()`	Move file pointer to a specific location	`fseek(fp, offset, SEEK_SET);`
`ftell()`	Get current file pointer position	`long pos = ftell(fp);`
`rewind()`	Move file pointer to the beginning	`rewind(fp);`

Detailed: fseek(), ftell(), fread(), fwrite()

fseek()

- Moves the file pointer to a specific location.
- **Syntax:**

```
int fseek(FILE *stream, long offset, int whence);
```

- `whence` can be `SEEK_SET` (beginning), `SEEK_CUR` (current), or `SEEK_END` (end)[8] [9].

Example:

```
fseek(fp, 10, SEEK_SET); // Move to 10th byte from start
```

ftell()

- Returns the current position of the file pointer.
- **Syntax:**

```
long ftell(FILE *stream);
```

Example:

```
long pos = ftell(fp);
printf("Current position: %ld\n", pos);
```

- Useful for finding file size:

```
fseek(fp, 0, SEEK_END);
long size = ftell(fp);
```

fread()

- Reads a block of binary data from a file.
- **Syntax:**

```
size_t fread(void *ptr, size_t size, size_t nmemb, FILE *stream);
```

- Reads `nmemb` elements, each of `size` bytes, into `ptr`[5] [6].

Example:

```
int arr[^5];
fread(arr, sizeof(int), 5, fp);
```

fwrite()

- Writes a block of binary data to a file.
- **Syntax:**

```
size_t fwrite(const void *ptr, size_t size, size_t nmemb, FILE *stream);
```

- Writes `nmemb` elements, each of `size` bytes, from `ptr` to file[5] [6].

Example:

```
int arr[^5] = {1,2,3,4,5};
fwrite(arr, sizeof(int), 5, fp);
```

Key Points & Differences

- **Text files** are for human-readable data; **binary files** are for efficient storage and processing of raw data[1] [2].
- Always check if `fopen()` returns `NULL` before proceeding.
- Use `fclose()` to properly close files and avoid memory leaks[3].
- Use `fseek()` and `ftell()` for random access within files.
- Use `fread()`/`fwrite()` for binary data; `fprintf()`/`fscanf()` for formatted text.

Let me know if you want sample programs or more details on any specific function!

⁂

1. https://byjus.com/gate/file-handling-in-c/
2. https://www.scaler.com/topics/c/file-handling-in-c/
3. https://www.w3schools.com/c/c_files.php
4. https://www.w3schools.com/c/c_files_write.php
5. https://www.w3resource.com/c-programming/c-file-handling.php
6. https://www.tutorialspoint.com/c_standard_library/c_function_fwrite.htm
7. https://www.w3schools.com/c/c_files_read.php
8. https://www.w3schools.com/c/ref_stdio_fseek.php
9. https://www.tutorialspoint.com/c_standard_library/c_function_fseek.htm