1. File Handling in C

Opening a File

To open a file, you can use the `fopen()` function, which returns a pointer to a `FILE` object.

Syntax:

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

- **`filename`**: The name of the file to open.
- **`mode`**: The mode in which to open the file, such as:
  - `"r"`: Read
  - `"w"`: Write (creates a new file or overwrites an existing file)
  - `"a"`: Append (creates a new file or appends to an existing file)
  - `"r+"`: Read and write
  - `"w+"`: Read and write (creates a new file or overwrites the existing file)
  - `"a+"`: Read and append

Example:

```
#include <stdio.h>

int main() {
    FILE *file = fopen("example.txt", "r");
    if (file == NULL) {
        perror("Failed to open file");
        return 1; // Exit if opening the file failed
    }
    // File operations...
    fclose(file); // Closing the file
    return 0;
}
```

Reading from a File

You can read from a file using various functions. Here are some common ones:

1. **`fgetc()`**: Reads a single character from a file.

   ```
   char c = fgetc(file);
   ```

2. **`fgets()`**: Reads a string from a file until a newline or EOF.

   ```
   char buffer[100];
   fgets(buffer, sizeof(buffer), file); // Reads a line from the file
   ```

3. **`fread()`**: Reads a block of data from a file.

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

   Example of using `fread()`:

```
int arr[5];
size_t elements_read = fread(arr, sizeof(int), 5, file);
```

Writing to a File

You can write to a file using these functions:

1. `fputc()` : Writes a single character to a file.

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

2. `fputs()` : Writes a string to a file.

   ```
   fputs("Hello, World!", file);
   ```

3. `fwrite()` : Writes a block of data to a file.

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

   Example of using `fwrite()` :

   ```
   int arr[5] = {1, 2, 3, 4, 5};
   fwrite(arr, sizeof(int), 5, file); // Writes 5 integers to the file
   ```

Closing a File

To release resources and avoid memory leaks, always close files after operations using `fclose()` :

```
int fclose(FILE *stream);
```

Example:

```
fclose(file);
```

Accessing Files

C provides the `access()` function (from `<unistd.h>`) to check the existence or accessibility of a file.

Syntax:

```
int access(const char *pathname, int mode);
```

- `pathname` : The name of the file.
- `mode` : Can be one of the following:
  - `F_OK` : Check for existence
  - `R_OK` : Check for readability
  - `W_OK` : Check for writability
  - `X_OK` : Check for executability

Example:

```
#include <unistd.h>
#include <stdio.h>
```

```c
int main() {
    if (access("example.txt", F_OK) != -1) {
        printf("File exists\n");
    } else {
        perror("File does not exist");
    }
    return 0;
}
```

Complete File Handling Example

Here’s an example demonstrating how to use some of these functions together:

```c
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>

int main() {
    // Check if the file exists
    if (access("example.txt", F_OK) == -1) {
        printf("Creating a new file...\n");
        FILE *file = fopen("example.txt", "w"); // Open for writing
        if (file == NULL) {
            perror("Error creating file");
            return 1;
        }

        // Write some data
        fputs("Hello, World!\n", file);
        fclose(file);
    } else {
        printf("File exists. Reading data...\n");

        FILE *file = fopen("example.txt", "r"); // Open for reading
        if (file == NULL) {
            perror("Error opening file");
            return 1;
        }

        char buffer[100];
        while (fgets(buffer, sizeof(buffer), file) != NULL) {
            printf("%s", buffer); // Print each line
        }

        fclose(file); // Close the file
    }

    return 0;
}
```

Summary

1. **File Operations**:
    - **Opening**: `fopen()`
    - **Reading**: `fgetc()`, `fgets()`, `fread()`
    - **Writing**: `fputc()`, `fputs()`, `fwrite()`
    - **Closing**: `fclose()`
2. **File Accessibility**:
    - Use `access()` to check if a file exists or check its permissions.
3. **Error Handling**: Always check for `NULL` when opening files and use `perror()` to handle errors.

1. `fopen()`

Syntax:

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

Return Value:

- **On success**: Returns a pointer to a `FILE` object that can be used to identify the file.
- **On failure**: Returns `NULL`. Use `perror()` or `strerror(errno)` to get the error description.

Example:

```
FILE *file = fopen("example.txt", "r");
if (file == NULL) {
    perror("Error opening file");
    // Handle the error
}
```

2. `fclose()`

Syntax:

```
int fclose(FILE *stream);
```

Return Value:

- **On success**: Returns `0`.
- **On failure**: Returns EOF (which is usually `-1`). You can check `errno` to find out the reason for the failure.

Example:

```
if (fclose(file) != 0) {
    perror("Error closing file");
    // Handle the error
}
```

3. `fgetc()`

Syntax:

```
int fgetc(FILE *stream);
```

Return Value:

- **On success**: Returns the next character as an `unsigned char` cast to an int or `EOF` on end-of-file or error.
- **On failure**: Returns `EOF`. You can check `feof()` and `ferror()` to distinguish between end-of-file and an error.

Example:

```
int ch = fgetc(file);
if (ch == EOF) {
    if (feof(file)) {
        printf("End of file reached.\n");
    } else {
        perror("Error reading character from file");
    }
}
```

4. `fgets()`

Syntax:

```
char *fgets(char *str, int n, FILE *stream);
```

Return Value:

- **On success**: Returns `str`, with the string read from the file.
- **On failure**: Returns `NULL`. This can also happen if the end-of-file is reached without reading any characters.

Example:

```
char buffer[100];
if (fgets(buffer, sizeof(buffer), file) == NULL) {
    if (feof(file)) {
        printf("End of file reached.\n");
    } else {
        perror("Error reading line from file");
    }
}
```

5. `fwrite()`

Syntax:

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

Return Value:

- **On success**: Returns the number of elements successfully written. If this number is less than `count`, it could indicate an error or that the end of the file was

reached.
- **On failure**: Returns a number less than `count`, and you should check `ferror()` to see if an error occurred.

Example:

```
size_t elements_written = fwrite(arr, sizeof(int), 5, file);
if (elements_written < 5) {
    if (ferror(file)) {
        perror("Error writing to file");
    } else {
        printf("Not all data was written to the file.\n");
    }
}
```

6. `fread()`

Syntax:

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

Return Value:

- **On success**: Returns the number of elements successfully read. If this number is less than `count`, it could indicate an error or that the end of the file was reached.
- **On failure**: Similar to `fwrite()`, it returns a number less than `count`, and you should check `ferror()`.

Example:

```
size_t elements_read = fread(arr, sizeof(int), 5, file);
if (elements_read < 5) {
    if (ferror(file)) {
        perror("Error reading from file");
    } else {
        printf("Not all data was read from the file or reached EOF.\n");
    }
}
```

7. `access()`

Syntax:

```
int access(const char *pathname, int mode);
```

Return Value:

- **On success**: Returns `0`. This means the file exists and is accessible in the requested mode (based on the value of `mode`).
- **On failure**: Returns `-1`, and you can use `perror()` or `strerror(errno)` to understand the reason (e.g., file not found, permission denied).

Example:

```c
if (access("example.txt", F_OK) == -1) {
    perror("File does not exist or cannot be accessed");
} else {
    printf("File exists and is accessible.\n");
}
```

Summary of Return Values

Function	Success Return Value	Failure Return Value
`fopen()`	Pointer to `FILE`	`NULL`
`fclose()`	`0`	`EOF` (typically `-1`)
`fgetc()`	Character (int)	`EOF`
`fgets()`	Pointer to string	`NULL`
`fwrite()`	Number of elements wrote	Number < count
`fread()`	Number of elements read	Number < count
`access()`	`0`	`-1`

Error Handling

It's essential to always check return values when performing file operations. Use the following functions for error handling:

- **`perror()`**: Prints a description for the last error that occurred in the file operations.
- **`ferror(FILE *stream)`**: Checks if a read/write error occurred.
- **`feof(FILE *stream)`**: Checks if the end of the file has been reached.