File handling in C programming allows you to create, read, write, and modify files from within a C program. Here’s a summary of the essential concepts and functions involved:

File Handling Functions in C

C provides several functions to manage file input and output. The most commonly used ones are:

1. `fopen()`:
   - Opens a file and associates it with a file pointer.
   - Syntax: `FILE *fopen(const char *filename, const char *mode);`
   - Modes:
     - `"r"`: Opens a file for reading (file must exist).
     - `"w"`: Opens a file for writing (creates a new file if not exists, truncates the file if exists).
     - `"a"`: Opens a file for appending (creates a new file if not exists).
     - `"r+"`: Opens a file for both reading and writing.
     - `"w+"`: Opens a file for both reading and writing (truncates the file if exists).
     - `"a+"`: Opens a file for both reading and appending.

2. `fclose()`:
   - Closes an opened file.
   - Syntax: `int fclose(FILE *stream);`

3. `fgetc()`:

- Reads a character from the file.
- Syntax: `int fgetc(FILE *stream);`

4. `fputc()`:
   - Writes a character to the file.
   - Syntax: `int fputc(int char, FILE *stream);`

5. `fgets()`:
   - Reads a string from a file.
   - Syntax: `char *fgets(char *str, int n, FILE *stream);`

6. `fputs()`:
   - Writes a string to a file.
   - Syntax: `int fputs(const char *str, FILE *stream);`

7. `fscanf()` and `fprintf()`:
   - `fscanf()`: Reads formatted input from a file.
     - Syntax: `int fscanf(FILE *stream, const char *format, ...);`
   - `fprintf()`: Writes formatted output to a file.
     - Syntax: `int fprintf(FILE *stream, const char *format, ...);`

8. `fseek()`:
   - Moves the file pointer to a specific location in the file.
   - Syntax: `int fseek(FILE *stream, long offset, int origin);`
   - `origin`:

- `SEEK_SET`: Beginning of the file.
- `SEEK_CUR`: Current position of the file pointer.
- `SEEK_END`: End of the file.

9. `ftell()`:
   - Returns the current position of the file pointer.
   - Syntax: `long ftell(FILE *stream);`

10. `rewind()`:
    - Sets the file pointer to the beginning of the file.
    - Syntax: `void rewind(FILE *stream);`

11. `feof()`:
    - Checks if the end of a file has been reached.
    - Syntax: `int feof(FILE *stream);`

12. `ferror()`:
    - Checks for any error in file operations.
    - Syntax: `int ferror(FILE *stream);`

Basic File Handling Example

```c
#include <stdio.h>

int main() {
```

```
FILE *fp;
char data[100];

// Opening a file in write mode
fp = fopen("file.txt", "w");
if (fp == NULL) {
    printf("Error opening file!\n");
    return 1;
}

// Writing to the file
fprintf(fp, "Hello, world!\n");
fclose(fp);

// Opening the same file in read mode
fp = fopen("file.txt", "r");
if (fp == NULL) {
    printf("Error opening file!\n");
    return 1;
}

// Reading from the file
fgets(data, 100, fp);
printf("Read from file: %s", data);
```

```
    fclose(fp);
    return 0;
}
```

File Modes Summary

Mode	Description
`"r"`	Open for reading (fails if the file doesn’t exist).
`"w"`	Open for writing (creates or truncates a file).
`"a"`	Open for appending (creates a file if it doesn’t exist).
`"r+"`	Open for reading and writing (fails if the file doesn’t exist).
`"w+"`	Open for reading and writing (creates or truncates a file).
`"a+"`	Open for reading and appending (creates a file if it doesn’t exist).

Error Handling in File Operations

1. Check for `NULL` when opening files to ensure the file was opened successfully.

```c
```

```
FILE *fp = fopen("example.txt", "r");
if (fp == NULL) {
    printf("Error opening file!\n");
}
```

2. Use `ferror()` to check if any errors occurred during file operations.

```
if (ferror(fp)) {
    printf("Error reading the file!\n");
}
```

Main points:

- Forgetting to close a file (`fclose`) after you are done using it.
- Not checking for errors while opening, reading, or writing to a file.
- Incorrect mode selection for the file (e.g., opening in read mode but trying to write).