File handling in c

File handling in C refers to the ability to work with files, including reading from and writing to files. It allows you to perform various operations on files, such as opening, closing, reading, writing, and modifying their contents.

In C, file handling is done through the use of file pointers and a set of functions provided by the standard C library, such as **fopen(), fclose(), fread(), fwrite(), fseek(),** and many others.

Here's a brief overview of the basic file handling operations in C:

1. Opening a File: To open a file, you use the fopen() function, which takes two arguments: the file name and the mode in which you want to open the file. The mode can be "r" for reading, "w" for writing (overwriting existing file or creating a new file), "a" for appending (writing at the end of an existing file), or a combination of these modes.

2. Closing a File: After you are done working with a file, you should close it using the fclose() function. It's important to close the file to release system resources associated with it.
3. Reading from a File: To read from a file, you use functions like fscanf() or fgets() to extract data from the file and store it in variables or character arrays.
4. Writing to a File: To write to a file, you can use functions like fprintf() or fputs() to write data from variables or character arrays to the file.
5. Moving within a File: You can use the fseek() function to move the file pointer to a specific position in the file. This allows you to read or write data at a particular location.
6. Error Handling: File operations can encounter errors, such as when a file does not exist or cannot be opened. You should check the return values of file operations for error conditions and handle them appropriately.
7. File Management: C provides functions like rename() and remove() to rename or delete files respectively. These are just the basic operations, and there are many more advanced file handling

functions and techniques available in C. It's important to handle errors properly and ensure that files are closed after use to avoid resource leaks.

**1. Create a file :**

```
// Step 1: Include the necessary header file
#include <stdio.h>

int main() {

    // Step 2:  Declare a file pointer variable
    FILE *filePtr;

    // Open the file in write mode
// Step 3: Open the file using fopen()
 filePtr = fopen("example.txt", "w");

    // Check if the file was created successfully
    if (filePtr == NULL) {
        printf("Unable to create the file.\n");
        return 1;
    }
```

```
    printf("File created successfully.\n");

    // Step 4: Close the file
    fclose(filePtr);

    return 0;
}
```

## 2. Writing data into the file:

```
#include <stdio.h>

int main() {
    FILE *filePtr;
    char data[100];

    // Open the file in write mode
    filePtr = fopen("example.txt", "w");

    // Check if the file was opened successfully
    if (filePtr == NULL) {
        printf("Unable to open the file.\n");
        return 1;
    }

    // Get input from the user
```

```
    printf("Enter data to write to the file: ");
    fgets(data, sizeof(data), stdin);

    // Write the data to the file
    fprintf(filePtr, "%s", data);

    // Close the file
    fclose(filePtr);

    printf("Data written to the file successfully.\n");

    return 0;
}
```

**3. Reading data from the file;**

```
#include <stdio.h>

int main() {
    FILE *filePtr;
    char data[100];

    // Open the file in read mode
    filePtr = fopen("example.txt", "r");

    // Check if the file was opened successfully
```

```
    if (filePtr == NULL) {
        printf("Unable to open the file.\n");
        return 1;
    }

    // Read data from the file
    fgets(data, sizeof(data), filePtr);

    // Close the file
    fclose(filePtr);

    // Display the read data
    printf("Data read from the file: %s\n", data);

    return 0;
}
```

**4. To delete a file:**

```
#include <stdio.h>

int main() {
    // Specify the file name to be deleted
    char fileName[] = "example.txt";

    // Attempt to delete the file
```

```
    int result = remove(fileName);

    // Check if the file deletion was successful
    if (result == 0) {
        printf("File deleted successfully.\n");
    } else {
        printf("Unable to delete the file.\n");
    }

    return 0;
}
```

5. To rename a file:

```
#include <stdio.h>

int main() {
    // Specify the old and new file names
    char oldName[] = "example.txt";
    char newName[] = "new_example.txt";

    // Attempt to rename the file
    int result = rename(oldName, newName);

    // Check if the file renaming was successful
    if (result == 0) {
```

```
        printf("File renamed successfully.\n");
    } else {
        printf("Unable to rename the file.\n");
    }

    return 0;
}
```