File Handling in C: A Beginner's Guide

Introduction

File handling in C allows you to create, read, write, and manipulate files. This tutorial covers the basics, such as accessing files, reading, writing, appending data, and searching for words in a file.

1 Accessing Files

File Pointer

Declare a file pointer using:

```
FILE *file_pointer;
```

Example:

```
FILE *fp;
```

Opening a File

Use `fopen()` to open a file in various modes:

- `"r"`: Open for reading (file must exist).
- `"w"`: Open for writing (creates a new file or overwrites if it exists).
- `"a"`: Open for appending (creates a new file if it doesn't exist).
- `"r+"`: Open for reading and writing.
- `"w+"`: Open for reading and writing (overwrites existing content).
- `"a+"`: Open for reading and appending.

Syntax:

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

Example:

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

Checking for Errors

If the file doesn't open, `fopen()` returns `NULL`.

```
if (fp == NULL) {
    printf("Error opening file!\n");
}
```

2 Reading from a File

Using `fgetc()`

Reads one character at a time:

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

Using `fgets()`

Reads a string (line) from the file:

```
char str[100];
fgets(str, 100, fp);
printf("%s", str);
```

Using `fscanf()`

Reads formatted input from a file:

```
int num;
fscanf(fp, "%d", &num);
printf("%d", num);
```

3 Writing to a File

Using `fputc()`

Writes a single character to a file:

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

Using `fputs()`

Writes a string to a file:

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

Using `fprintf()`

Writes formatted data to a file:

```
fprintf(fp, "Age: %d\n", 25);
```

4 Appending Data

Open the file in `"a"` or `"a+"` mode and use the same writing functions.

5 Closing a File

Use `fclose(fp)` to close the file after operations. This ensures all data is written and resources are freed.

6 Searching for a Word in a Paragraph

To search for a word in a paragraph, follow these steps:

- Read the file line by line.
- Use `strstr()` to check if the word exists in each line.
- Keep a counter for occurrences.

Example Code

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

int main() {
    FILE *fp = fopen("paragraph.txt", "r");
    char line[256];
    char word[] = "example";
    int count = 0;

    if (fp != NULL) {
        while (fgets(line, sizeof(line), fp)) {
            char *ptr = strstr(line, word);
            while (ptr != NULL) {
                count++;
                ptr = strstr(ptr + 1, word);
            }
        }
        fclose(fp);

        printf("The word '%s' was found %d times in the file.\n", word, count);
    } else {
```

```
        printf("Error opening file.\n");
    }
    return 0;
}
```

7 Enhancements for Searching

Case-Insensitive Search

Convert both the word and the line to lowercase before searching. Use `tolower()` from `<ctype.h>`.

Word Boundary Matching

Ensure the match is a whole word by checking surrounding characters using `isalpha()`.

8 Counting Words in the Entire File

Algorithm

Read the file word by word using `fscanf()` and increment the counter for each match.

Example Code

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

int main() {
    FILE *fp = fopen("paragraph.txt", "r");
    char word_to_find[] = "example";
    char word[50];
    int count = 0;

    if (fp != NULL) {
        while (fscanf(fp, "%s", word) == 1) {
            if (strcmp(word, word_to_find) == 0) {
                count++;
            }
        }
        fclose(fp);

        printf("The word '%s' was found %d times in the file.\n", word_to_find, count
    } else {
        printf("Error opening file.\n");
    }
    return 0;
}
```

9 Conclusion

This guide covers the basics of file handling, including creating, reading, writing, appending, and searching for words in files. By mastering these techniques, you can efficiently handle file-based operations in C.