C File Handling

File Handling Basics

A large program deployed for heavy applications cannot function without files, since we can get input from them as well as print output from them very easily. We can also save a lot of program space by accessing the file's data only when needed, making the program more efficient and faster.

Why do we need to handle files in C?

- Files are used to store content hence reducing the actual program's size.
- We can read or access data from files.
- The data in files remain stored even after the program's execution is terminated.

Files are stored in non-volatile memory. To understand what a non-volatile memory is and how it is better in terms of storing things for longer, we have to see the **differences between volatile and non-volatile memory.**

What are the different types of files?

There are two types of files:

- **Binary Files**

Binary files store data in 01 i.e., binary format. They are not directly readable. In order to read binary files, you will need software or an application. An example of a binary file is a .bin file.

- **Text Files**

Text files store data in a simple text format. They are directly readable and no external software is required to access them. An example of a text file is a .txt file.

Operations on Files

There are basically four operations we can perform on files in C:

- **Creating a File:**

We can create a file using C language, in any directory, without even leaving our compiler. We can select the name or type we want our file to have, along with its location.

- **Opening a File:**

We can open an existing file and create a new file and open it using our program. We can perform different operations on a file after it has been opened.

- **Closing a File:**

When we are done with the file, meaning that we have performed whatever we want to perform on our file, we can close the file using the close function.

- **Read/Write to a file:**

After opening a file, we can access its contents and read, write, or update them.

Files I/O

The first and foremost thing we should know when working with files in C is that we have to declare a pointer of the file type to work with files. The syntax for declaring a pointer of file type is:

```
FILE *ptr;
```

Modes

Functions and their modes of declarations are two important factors of file handling. We have to learn about different modes used along with these functions as a parameter. The following are the modes:

- r: opens a file for reading.
- w: opens a file for writing. It can also create a new file.
- a: opens a file for appending.
- r+: opens a file for both reading and writing but cannot create a new file.
- w+: opens a file for both reading and writing.

There are many other modes, but these are the basic and most used ones.

Closing a file

When working with C, closing open files is an essential step. A programmer often makes a mistake of not closing an open file. This becomes crucial because files do not automatically get closed after a program uses them. The closing has to be done manually.

To close a file, we have to use the fclose() function. We only need to pass the pointer as a parameter to the function.

Syntax:

```
fclose(ptr);
```

Reading a file

Reading from a file is as easy as reading any other stuff as an input in C. We just use a file version of scanf(). In order to read from a file, we use the function fscanf().
Like scanf() used to get input from the keyboard, it gets its input from a file and prints it onto the screen.

We have to send the file pointer as an argument for the program to be able to read it. The file has to be opened in r mode, i.e., read mode, to work properly for fscanf().

Example:

```
#include <stdio.h>


int main()

{

   FILE *ptr;

   ptr = fopen("example.txt", "r");

   char str[128];

   fscanf(ptr, "%s", str);

   printf("%s", str);

}
```

The file example.txt had "Welcome_To_CodeWithHarry!" as its content, hence the output:

```
Welcome_To_CodeWithHarry!
```

Writing to a file

Writing to a file is as easy as printing any other stuff in C. We just use a file version of printf(). In order to write to a file, we use the function fprintf(). For printing text inside the file, we use fprintf() as we did for printing text on the screen using printf(). We have to send the file pointer as an argument for the program to be able to print it into the file. The file has to be opened in w mode, i.e., write mode, to be able to write in the file properly.

fprintf() takes the pointer to the file as one of the arguments along with the text to be written.

Example:

```
#include <stdio.h>


int main()
{
   FILE *ptr;
   ptr = fopen("example.txt", "w");
   char str[128] = "Hello World!";
   fprintf(ptr, "%s", str);
}
```

Output in the example.txt file:

```
Hello World!
```