File Handling

Handling in C

File handling refers to the method of storing data in the C program in the form of an output or input that might have been generated while running a C program in a data file, i.e., a binary file or a text file for future analysis and reference in that very program.

File in C

A file refers to a source in which a program stores the information/data in the form of bytes of sequence on a disk (permanently). The content available on a file isn't volatile like the compiler memory in C. But the program can perform various operations, such as creating, opening, reading a file, or even manipulating the data present inside the file. This process is known as file handling in C.

Why Do We Need File Handling in C?

There are times when the output generated out of a program after its compilation and running do not serve our intended purpose. In such cases, we might want to check the program's output various times. Now, compiling and running the very same program multiple times becomes a tedious task for any programmer. It is exactly where file handling becomes useful.

Let us look at a few reasons why file handling makes programming easier for all:

- **Reusability:** File handling allows us to preserve the information/data generated after we run the program.
- **Saves Time:** Some programs might require a large amount of input from their users. In such cases, file handling allows you to easily access a part of a code using individual commands.
- **Commendable storage capacity:** When storing data in files, you can leave behind the worry of storing all the info in bulk in any program.
- **Portability:** The contents available in any file can be transferred to another one without any data loss in the computer system. This saves a lot of effort and minimises the risk of flawed coding.

Types of Files in a C Program

When referring to file handling, we refer to files in the form of data files. Now, these data files are available in 2 distinct forms in the C language, namely:

- Text Files
- Binary Files

Text Files

The text files are the most basic/simplest types of files that a user can create in a C program. We create the text files using an extension .txt with the help of a simple text editor. In general, we can use notepads for the creation of .txt files. These files store info internally in ASCII character format, but when we open these files, the content/text opens in a human-readable form.

Text files are, thus, very easy to access as well as use. But there's one major disadvantage; it lacks security. Since a .txt file can be accessed easily, information isn't very secure in it. Added to this, text files consume a very large space in storage.

To solve these problems, we have a different type of file in C programs, known as binary files.

Binary Files

The binary files store info and data in the binary format of 0's and 1's (the binary number system). Thus, the files occupy comparatively lesser space in the storage. In simpler words, the binary files store data and info the same way a computer holds the info in its memory. Thus, it can be accessed very easily as compared to a text file.

The binary files are created with the extension .bin in a program, and it overcomes the drawback of the text files in a program since humans can't read it; only machines can. Thus, the information becomes much more secure. Thus, binary files are safest in terms of storing data files in a C program.

Operators/Functions that We Use for File Handling in C

We can use a variety of functions in order to open a file, read it, write more data, create a new file, close or delete a file, search for a file, etc. These are known as file handling operators in C.

Here's a list of functions that allow you to do so:

Description of Function	**Function in Use**
used to open an existing file or a new file	fopen()
writing data into an available file	fprintf()
reading the data available in a file	fscanf()
writing any character into the program file	fputc()
reading the character from an available file	fgetc()
used to close the program file	fclose()
used to set the file pointer to the intended file position	fseek()
writing an integer into an available file	fputw()

used to read an integer from the given file	fgetw()
used for reading the current position of a file	ftell()
sets an intended file pointer to the file's beginning itself	rewind()

Note: It is important to know that we must declare a file-type pointer when we are working with various files in a program. This helps establish direct communication between a program and the files.

Here is how you can do it:

FILE *fpointer;

Out of all the operations/functions mentioned above, let us discuss some of the basic operations that we perform in the C language.

Operations Done in File Handling

The process of file handling enables a user to update, create, open, read, write, and ultimately delete the file/content in the file that exists on the C program's local file system. Here are the primary operations that you can perform on a file in a C program:

- Opening a file that already exists
- Creating a new file
- Reading content/ data from the existing file
- Writing more data into the file
- Deleting the data in the file or the file altogether

Opening a File in the Program – to create and edit data

We open a file with the help of the fopen() function that is defined in the header file- stdio.h.

Here is the syntax that we follow when opening a file:

ptr = fopen ("openfile" , "openingmode");

Let us take a look at an example for the same,

fopen ("E:\\myprogram\\recentprogram.txt" , "w");

fopen ("E:\\myprogram\\previousprogram.bin" , "rb");

- Here, if we suppose that the file – recentprogram.txt doesn't really exist in the E:\\myprogram location. Here, we have used the mode "w". Thus, the first function will create a new file with the name recentprogram.txt and then open it for writing (since we have used the "w" mode).
- The "w" here refers to writing mode. It allows a programmer to overwrite/edit and create the contents in a program file.

- Now, let us take a look at the second binary previousprogram.bin file that is present in the E:\\myprogram location. Thus, the second function here will open the file (that already exists) for reading in the “rb” binary mode.
- The “rb” refers to the reading mode. It only allows you to read a file, but not overwrite it. Thus, it will only read this available file in the program.

Let us take a look at a few more opening modes used in the C programs:

Opening Modes of C in Standard I/O of a Program		
Mode in Program	**Meaning of Mode**	**When the file doesn’t exist**
r	Open a file for reading the content.	In case the file doesn’t exist in the location, then fopen() will return NULL.
rb	Open a file for reading the content in binary mode.	In case the file doesn’t exist in the location, then fopen() will return NULL.
w	Open a file for writing the content.	In case the file exists, its contents are overwritten. In case the file doesn’t exist in the location, then it will create a new file.
wb	Open a file for writing the content in binary mode.	In case the file exists, then its contents will get overwritten. In case the file doesn’t exist in the location, then it will create a new file.
a	Open a file for appending the content. Meaning, the data of the program is added to the file’s end in a program.	In case the file doesn’t exist in the location, then it will create a new file.
ab	Open a file for appending the content in binary mode. Meaning, the data of the program is added to the file’s end in a program in a binary mode.	In case the file doesn’t exist in the location, then it will create a new file.
r+	Open a file for both writing and reading the	In case the file doesn’t exist in the

	content.	location, then fopen() will return NULL.
rb+	Open a file for both writing and reading the content in binary mode.	In case the file doesn't exist in the location, then fopen() will return NULL.
w+	Open a file for both writing and reading.	In case the file exists, its contents are overwritten. In case the file doesn't exist in the location, then it will create a new file.
wb+	Open a file for both writing and reading the content in binary mode.	In case the file exists, its contents are overwritten. In case the file doesn't exist in the location, then it will create a new file.
a+	Open a file for both appending and reading the content.	In case the file doesn't exist in the location, then it will create a new file.
ab+	Open a file for both appending and reading the content in binary mode.	In case the file doesn't exist in the location, then it will create a new file.

How do we close a file?

Once we write/read a file in a program, we need to close it (for both binary and text files). To close a file, we utilise the fclose() function in a program.

Here is how the program would look like:

fclose(fptr);

In this case, the fptr refers to the file pointer that is associated with that file that needs to be closed in a program.

How do we read and write the data to the text file?

We utilise the fscanf() and fprintf() to write and read the data to the text file. These are basically the file versions of the scanf() and printf(). But there is a major difference, i.e., both fscanf() and fprintf() expect a pointer pointing towards the structure FILE in the program.

Example #1: Writing Data to the Text File in a Program

#include <stdio.h>

#include <stdlib.h>

```
int main()

{

int val;

FILE *fptr;

// if you are using Linux or MacOS, then you must use appropriate locations

fptr = fopen (“C:\\currentprogram.txt”,”w”);

if(fptr == NULL)

{

printf(“File type invalid!”);

exit(1);

}

printf(“Please enter the val: “);

scanf(“%d”,&val);

fprintf(fptr,”%d”,val);

fclose(fptr);

return 0;

}
```

The program mentioned here would take the number given by then store it in the currentprogram.txt file.

Once we compile this program and run it on the system, we will be able to witness a currentprogram.txt text file created in the C drive of our computer! Also, when we open this file, then we will be able to see what integer we entered as an input while coding.

Example #2: Reading Information from the Text File in a Program

```
#include <stdio.h>

#include <stdlib.h>

int main()

{

int val;

FILE *fptr;

// The program will exit in case the file pointer fptr returns NULL.

if ((fptr = fopen(“C:\\currentprogram.txt”,”r”)) == NULL){

printf(“Visible error detected. Cannot open the file!”);
```

```
exit(1);

}

fscanf(fptr,"%d", &val);

printf("The value of the integer n is=%d", val);

fclose(fptr);

return 0;

}
```

The program given here would read the integer that we have input in the currentprogram.txt file and then print the integer as an output on the computer screen.

In case you were successful in creating the file with the help of Example 1, then the running of the program in Example 2 will generate the integer that you have input.

You can also use other functions such as fputc(), fgetchar(), etc., in a similar manner in the program.

How do we Read and Write the Data to the Binary File in a Program?

We have to use the functions fwrite() and fread() for writing to and reading from a file present on a disk respectively, for the binary files.

When Writing to the Binary File

We utilise the function fwrite() for writing the data into a binary file in the program. This function would primarily take four arguments:

- the address of the data that would be written on the disk
- the size of this data
- total number of such types of data on the disk
- the pointers to those files where we would be writing

It would go something in this line:

```
fwrite (address_of_Data, size_of_Data, numbers_of_Data, pointerToFile);
```

When Reading from the Binary File

Just like the above-used function fwrite(), the function fread() also takes 4 arguments (lik above).

It would go something like this:

```
fread (address_of_Data, size_of_Data, numbers_of_Data, pointerToFile);
```