File Handling in C

In this tutorial, you will master all about C files and the operations with examples. You will cover what a file is, how to open and close a file, and some basic ways to read and write files through C programming.

What is a File?

A file is a collection of bytes with specific names. Precisely, A file is a named location on the system storage that stores related data for future purposes. The data can be anything like a simple text file, video or audio file, or any complex executable program. We use the file system to enable persistent storage in non-volatile memory like the hard disk.

Why file in C?

When we compile and run a program, we definitely get an output. As soon as we exit the program, all the input and output get erased from the memory. I/O functions help us store these data in the hard disk or removable disks in the form of a file thereby ensuring the permanent storage of data. The other reason is that files are easy when handling large data as entering large data into a program is not at all considered a fun practice. Data stored in a file can be accessed using file commands. Moreover, files are easy to transfer between computers.

Types of files

Files are classified into two types :

1. **Binary Files** are files that contain non-text values like images, audios, etc. Usually, binary files contain objects in the form of zeros and ones. Binary files are **.bin** files in your computer which are not user-friendly and so are not readable but provide high security.
2. **Text Files** are files that contain text values that are structured into lines. A text file, in short, contains multiple lines of text values. Each line ends with a special character referred to as End of Line and usually saved with **.txt** files.

File Operations in C

Below listed are the 4 basic operations one needs to perform when working with files. They are listed in the order of processing:

1. Create a new file
2. Open an existing file
3. Read or Write operation
4. Close an opened file

Flowchart for file handling

How to open a file?

As files remain stored on the disk, we need to create a link between the file and the program by declaring a pointer of the type file as given below.

```
FILE *fptr;
```

Then we can open a file first in order to read or write it. **fopen()** function is generally used to do this job. It tells the operating system to open the file name and the reason for opening it as well. If the operating system succeeds to do so, it sends the address of the first byte of the file as a pointer to the structure file. The prototype of this function is :

```
FILE *fopen (const char *filename, const char *mode);
```

where

- **filename** gives the name of the file to be opened,
- **mode** represents the file's opening mode (e.g. r- read, write, append).

Suppose we want to open a file **'Data.dat'** for reading purposes. In this case, the arguments will be:

```
FILE *fptr;
fptr= fopen("Data.dat", "r");
```

Here 'r' denotes that the file will be read-only in nature. Clearly, the second parameter of the argument determines the property of the file. Here is the table showing different opening modes of files.

Modes	Meaning	Description
r	Read	Opens a text file for reading. **fopen()** returns Null if the file does not exist.
w	Write	Opens text file in write-only mode to overwrite. Creates a file if it does not exist
a	Append	Appends the file at the end of the file without trimming the existing. Creates a file if it does not exist.
r+	Read and Write	Opens file for both reading and writing
w+	Read and write	Opens file in writing and reading mode
a+	Append read and write	Open file in appending mode for reading and writing
rb	Binary read	Opens a binary file for reading
wb	Binary write	Opens a binary file for writing
ab	Binary append	Opens a binary file in append at the end of the file
rb+	Binary read and write	Opens a binary file for reading and writing
wb+	Binary read and write	Opens a binary file for writing and reading
ab+	Binary append read and write	Opens a binary file in the read-write mode for appending

How to close a file?

After reading or writing, we must close it and the **fclose()** function does the job. The prototype of the function is mentioned in the **stdio.h** header file:

```
fclose(fp);
```

This function uses **fp**, a pointer to FILE structure, exactly similar to the **fopen** function. When we write data in a file, it is stored in a buffer initially. When the buffer becomes full or the compiler encounters a **fclose()** function, the data is actually written to the disk.

How to read from and write to a file?

Several functions are available to read from and write to data in a file. Below are the functions tabularised for easy understanding.

ID no.	Function	Description
1	fprintf()	writes data into the file
2	fscanf()	reads data from the file
3	fputc()	writes a character into the file
4	fgetc()	reads a character from the file
5	fseek()	sets the file pointer to a specific position
6	fputs()	writes a string to the file
7	fgets()	reads a string from the file
8	ftell()	returns current position
9	rewind()	sets the file pointer at the beginning
10	fread()	reads data from the binary file
11	fwrite()	writes data to the binary file

writing to a text file

To write something to a text file, simplest of all is **putc**. Its prototype looks like this:

```
putc (int ch, FILE *stream);
```

Here is an example, illustrating the application of this function.

```
#include <stdio.h>
main()
{
   FILE *fp;
   char ch;
   fp= fopen ("FILE.txt", "w");
   while ((ch=getchar()) != '\n')
      putc (ch, fp);
}
```

In this program, **getchar** will take single character input and store it in the variable **ch**. **putc** will use the file pointer **fp** to write that character in the target file **FILE.txt**.

fputs is another function, capable of writing a string in a file at once. Its syntax is

```
fputs (const char *str, FILE *stream);
```

Reading from text file

fgetc function is the simplest tool to read data inside a file. The prototype of this function is

```
fgetc(FILE *stream);
```

We can use a 'while loop' to read multiple characters by it.

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

Just like 'fputs', there is a function called 'fgets' if the input stream reference has 'n' number of characters, then it will be able to read (n-1) number of characters.

Example of reading from a text file

Suppose that we have a file as shown below with the name myfile.txt and contains some data.

Let us write a simple c program to read the file using **fgetc**.

```
#include<stdio.h>
void main()
{
   FILE *fp;
   char c;
   fp=fopen("myfile.txt","r");
   while((c=fgetc(fp))!=EOF)
   printf("%c",c);
   fclose(fp);
}
```

This program outputs the data contained in the file myfile.txt and the output printed on the console is

```
This is how we read data in a file using C programming.
```

Binary I/O functions

Till now we have discussed about text files. But we can also read, write or edit binary files through C programming. In this case the second parameter is 'rb' instead of 'r'. Similarly, 'w' is replaced by 'wb'.

```
fp= fopen ("Binary.bin", "rb");
```

In the case of text files, 'n' is regarded as a single character, whereas it is considered as two consecutive characters in binary files. Also, we can not add any 'end of file' character at the end of binary files.

'fread' and 'fwrite' are the basic functions, mostly used in reading or writing reading this kind of files. Their prototypes are

```
size_t fwrite (const *buf,  int size,  int, count,  FILE *fp);
```

and

```
size_t fread (void *buf,  int size,  int, count,  FILE *fp);
```

- **Buffer** : in short but, is a pointer which contains the address where data will be stored. A buffer is a memory location where data is stored temporarily.
- **Size** :The size of each element to be read or written in bytes.
- **Count** :Number of elements to be read or written
- **Stream**: Pointer to the FILE object from where data is to be read or written.

Examine the below program which writes the structure

```
#include<stdio.h>
#include<stdlib.h>
struct Book
{
   char Title[20];
   int price;
};
int main()
{
   struct Book B;
   FILE *fp;
   fp=fopen("Book.txt","wb");
   if(fp==NULL)
   {
      printf("Error: Cannot open file");
      exit(1);
   }
   printf("\n Enter the title of the book:");
   gets(B.Title);
   printf("\n Enter the price of the book:");
   fflush(stdin);
   scanf("%d",&B.price);
   fwrite(&B,sizeof(struct Book),1,fp);
   printf("\nData is stored in th file successfully in binary format");
   fclose(fp);
   return 0;
}
```

Output:

```
Enter the title of the book:Alchemist
Enter the price of the book:1000
Data is stored in the file successfully in binary format
```

In the above program we have created a structure Book with Title and Price as its members. In the main function we have declared a file pointer, fp and opened the Book.txt file to write the Title and price of the books. But we need to store the data in binary format so we make use of fwrite() function. Afterwards you can traverse to the file location and check what details are stored inside the file. Since the details are stored in binary format you can only read a few datas.

In order to read datas in a binary format you can use the fread() function as shown in the program below

```
#include<stdio.h>
#include<stdlib.h>
struct Book
{
   char Title[20];
   int price;
};
int main()
{
   struct Book B;
   FILE *fp;
   fp=fopen("Book.txt","rb");
   if(fp==NULL)
   {
      printf("Error: Cannot open file");
      exit(1);
   }
   fread(&B,sizeof(struct Book),1,fp);
   printf("\nTITLE OF BOOK : %s",B.Title);
   printf("\nPRICE OF BOOK : %d",B.price);
   fclose(fp);
}
```