

Files I/O

Why files are needed?

- When a program is terminated, the entire data is lost. Storing in a file will preserve your data even if the program terminates.
- If you have to enter a large number of data, it will take a lot of time to enter them all. However, if you have a file containing all the data, you can easily access the contents of the file using few commands in C.
- You can easily move your data from one computer to another without any changes.

Types of Files

When dealing with files, there are two types of files you should know about:

1. Text files
2. Binary files

1. Text files

Text files are the normal .txt files that you can easily create using Notepad or any simple text editors.

When you open those files, you'll see all the contents within the file as plain text. You can easily edit or delete the contents.

They take minimum effort to maintain, are easily readable, and provide least security and takes bigger storage space.

2. Binary files

Binary files are mostly the .bin files in your computer.

Instead of storing data in plain text, they store it in the binary form (0's and 1's).

They can hold higher amount of data, are not readable easily and provides a better security than text files.

File Operations

In C, you can perform four major operations on the file, either text or binary:

1. Creating a new file
2. Opening an existing file
3. Closing a file
4. Reading from and writing information to a file
5. Deleting file

Functions for file handling

No.	Function	Description
1	<i>fopen()</i>	<i>opens new or existing file</i>
2	<i>fprintf()</i>	<i>write data into the file</i>
3	<i>fscanf()</i>	<i>reads data from the file</i>
4	<i>fputc()</i>	<i>writes a character into the file</i>
5	<i>fgetc()</i>	<i>reads a character from file</i>
6	<i>fclose()</i>	<i>closes the file</i>
7	<i>fseek()</i>	<i>sets the file pointer to given position</i>
8	<i>fputw()</i>	<i>writes an integer to file</i>
9	<i>fgetw()</i>	<i>reads an integer from file</i>
10	<i>ftell()</i>	<i>returns current position</i>
11	<i>rewind()</i>	<i>sets the file pointer to the beginning of the file</i>

Working with files

When working with files, you need to declare a pointer of type file. This declaration is needed for communication between the file and program.

```
FILE *fptr;
```

Opening a file - for creation and edit

Opening a file is performed using the library function in the "**stdio.h**" header file: `fopen()`.

The syntax for opening a file in standard I/O is:

```
ptr = fopen("fileopen","mode")
```

File Mode	Meaning of Mode	During Inexistence of file
r	Open for reading.	If the file does not exist, fopen() returns NULL.
w	Open for writing.	If the file exists, its contents are overwritten. If the file does not exist, it will be created.
a	Open for append. i.e, Data is added to end of file.	If the file does not exist, it will be created.
r+	Open for both reading and writing.	If the file does not exist, fopen() returns NULL.
w+	Open for both reading and writing.	If the file exists, its contents are overwritten. If the file does not exist, it will be created.
a+	Open for both reading and appending.	If the file does not exist, it will be created.
rb	Open for reading in binary mode.	If the file does not exist, fopen() returns NULL.
wb	Open for writing in binary mode.	If the file exists, its contents are overwritten. If the file does not exist, it will be created.
ab	Open for append in binary mode. i.e, Data is added to end of file.	If the file does not exist, it will be created.
rb+	Open for both reading and writing in binary mode.	If the file does not exist, fopen() returns NULL.
wb+	Open for both reading and writing in binary mode.	If the file exists, its contents are overwritten. If the file does not exist, it will be created.
ab+	Open for both reading and appending in binary mode.	If the file does not exist, it will be created.

Reading and writing to a text file

For reading and writing to a text file, we use the functions `fprintf()` and `fscanf()`.

They are just the file versions of `printf()` and `scanf()`. The only difference is that, `fprint` and `fscanf` expects a pointer to the structure `FILE`.

Write to a text file using `fprintf()`

F1.c

This program takes a number from user and stores in the file `program.txt`.

After you compile and run this program, you can see a text file `program.txt` created in C drive of your computer. When you open the file, you can see the integer you entered.

Read from a text file using `fscanf()`

read.c

This program reads the integer present in the `program.txt` file and prints it onto the screen.

If you successfully created the file from **Example 1**, running this program will get you the integer you entered.

Other functions like `fgetchar()`, `fputc()` etc. can be used in similar way.

Reading and writing to a binary file

Functions `fread()` and `fwrite()` are used for reading from and writing to a file on the disk respectively in case of binary files.

Writing to a binary file

To write into a binary file, you need to use the function `fwrite()`. The function takes four arguments: Address of data to be written in disk, Size of data to be written in disk, number of such type of data and pointer to the file where you want to write.

```
fwrite(address_data, size_data, numbers_data, pointer_to_file);
```

Write to a binary file using fwrite()

writingB.c

In this program, you create a new file `program.bin` in the C drive.

We declare a structure `threeNum` with three numbers - *n1*, *n2* and *n3*, and define it in the main function as *num*.

Now, inside the for loop, we store the value into the file using `fwrite()`.

The first parameter takes the address of *num* and the second parameter takes the size of the structure `threeNum`.

Since, we're only inserting one instance of *num*, the third parameter is 1. And, the last parameter `*fptr` points to the file we're storing the data.

Finally, we close the file.

Reading from a binary file

Function `fread()` also take 4 arguments similar to `fwrite()` function as above.

```
fread(address_data, size_data, numbers_data, pointer_to_file);
```

Read from a binary file using fread()

readingB.c

In this program, you read the same file `program.bin` and loop through the records one by one.

In simple terms, you read one `threeNum` record of `threeNum` size from the file pointed by `*fptr` into the structure *num*.

Getting data using fseek()

If you have many records inside a file and need to access a record at a specific position, you need to loop through all the records before it to get the record.

This will waste a lot of memory and operation time. An easier way to get to the required data can be achieved using fseek().

As the name suggests, fseek() seeks the cursor to the given record in the file.

Syntax of fseek()

```
fseek(FILE * stream, long int offset, int whence)
```

The first parameter stream is the pointer to the file. The second parameter is the position of the record to be found, and the third parameter specifies the location where the offset starts

Different Whence in fseek	
Whence	Meaning
SEEK_SET	Starts the offset from the beginning of the file.
SEEK_END	Starts the offset from the end of the file.
SEEK_CUR	Starts the offset from the current location of the cursor in the file