

# C File Handling

A file is a container in computer storage devices used for storing data.

---

## 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 a few commands in C.

- You can easily move your data from one computer to another without any changes.

---

## File Operations

In C, you can perform four major operations on files, 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
-

# 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 the program.

```
FILE *fptr;
```

---

## Opening a file - for creation and edit

Opening a file is performed using the `fopen()` function defined in the `stdio.h` header file.

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

```
ptr = fopen("filename", "mode");
```

For example,

```
fopen("E:\\cprogram\\newprogram.txt", "w");
```

```
fopen("E:\\cprogram\\oldprogram.bin", "rb");
```

- Let's suppose the file `newprogram.txt` doesn't exist in the location `E:\\cprogram`. The first function creates a new file named `newprogram.txt` and opens it for writing as per the mode 'w'.  
The writing mode allows you to create and edit (overwrite) the contents of the file.
- Now let's suppose the second binary file `oldprogram.bin` exists in the location `E:\\cprogram`. The second function opens the existing file for reading in binary mode 'rb'.

The reading mode only allows you to read the file, you cannot write into the file.

Mode	Meaning of Mode	During Inexistence of file
r	Open for reading.	If the file does not exist, <code>fopen()</code> returns NULL.
rb	Open for reading in binary mode.	If the file does not exist, <code>fopen()</code> returns NULL.
w	Open for writing.	If the file exists, its contents are overwritten.  If the file does not exist, it will be created.
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.

a	Open for append.	If the file does not exist, it will be created.
	Data is added to the end of the file.	
ab	Open for append in binary mode.	If the file does not exist, it will be created.
	Data is added to the end of the file.	
r+	Open for both reading and writing.	If the file does not exist, <code>fopen()</code> returns NULL.
rb+	Open for both reading and writing in binary mode.	If the file does not exist, <code>fopen()</code> 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.

<code>wb+</code>	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.
<code>a+</code>	Open for both reading and appending.	If the file does not exist, it will be created.
<code>ab+</code>	Open for both reading and appending in binary mode.	If the file does not exist, it will be created.

## Closing a File

The file (both text and binary) should be closed after reading/writing.

Closing a file is performed using the `fclose()` function.

```
fclose(fptr);
```

Here, `fptr` is a file pointer associated with the file to be closed.

## 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 `fprintf()` and `fscanf()` expects a pointer to the structure `FILE`.

---

## Example 1: Write to a text file

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

int main()
{
    int num;
    FILE *fptr;

    // use appropriate location if you are using MacOS or Linux
    fptr = fopen("C:\\\\program.txt", "w");

    if(fptr == NULL)
    {
        printf("Error!");
        exit(1);
    }

    printf("Enter num: ");
    scanf("%d", &num);

    fprintf(fptr, "%d", num);
    fclose(fptr);

    return 0;
}
```

This program takes a number from the 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.

---

## Example 2: Read from a text file

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

int main()
{
    int num;
    FILE *fptr;

    if ((fptr = fopen("C:\\program.txt","r")) == NULL) {
        printf("Error! opening file");

        // Program exits if the file pointer returns NULL.
        exit(1);
    }

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

    printf("Value of n=%d", num);
    fclose(fptr);

    return 0;
}
```

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 a similar way.

---

---

## 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.

Whence	Meaning
<code>SEEK_SET</code>	Starts the offset from the beginning of the file.
<code>SEEK_END</code>	Starts the offset from the end of the file.
<code>SEEK_CUR</code>	Starts the offset from the current location of the cursor in the file.

---

## Example 5: `fseek()`



```

#include <stdio.h>
#include <stdlib.h>

struct threeNum
{
    int n1, n2, n3;
};

int main()
{
    int n;
    struct threeNum num;
    FILE *fptr;

    if ((fptr = fopen("C:\\program.bin", "rb")) == NULL) {
        printf("Error! opening file");

        // Program exits if the file pointer returns NULL.
        exit(1);
    }

    // Moves the cursor to the end of the file
    fseek(fptr, -sizeof(struct threeNum), SEEK_END);

    for(n = 1; n < 5; ++n)
    {
        fread(&num, sizeof(struct threeNum), 1, fptr);
        printf("n1: %d\tn2: %d\tn3: %d\n", num.n1, num.n2, num.n3);
        fseek(fptr, -2*sizeof(struct threeNum), SEEK_CUR);
    }
    fclose(fptr);

    return 0;
}

```

This program will start reading the records from the file `program.bin` in the reverse order (last to first) and prints it.