File Management in C

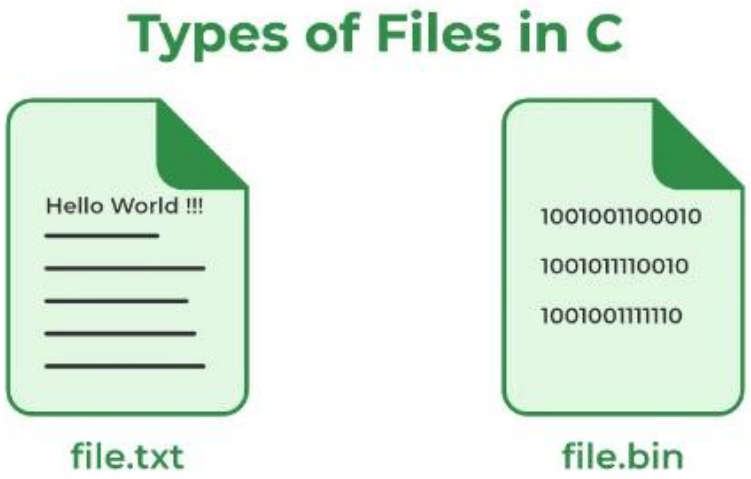
In order to understand why file handling is important, let us look at a few features of using files:

- **Reusability:** The data stored in the file can be accessed, updated, and deleted anywhere and anytime providing high reusability.
- **Portability:** Without losing any data, files can be transferred to another in the computer system. The risk of flawed coding is minimized with this feature.
- **Efficient:** A large amount of input may be required for some programs. File handling allows you to easily access a part of a file using few instructions which saves a lot of time and reduces the chance of errors.
- **Storage Capacity:** Files allow you to store a large amount of data without having to worry about storing everything simultaneously in a program.

Types of Files in C

A file can be classified into two types based on the way the file stores the data. They are as follows:

- Text Files
- Binary Files



1. Text Files

A text file contains data in the **form of ASCII characters** and is generally used to store a stream of characters.

- Each line in a text file ends with a new line character ('\n').
- It can be read or written by any text editor.
- They are generally stored with **.txt** file extension.
- Text files can also be used to store the source code.

2. Binary Files

A binary file contains data in **binary form (i.e. 0's and 1's)** instead of ASCII characters. They contain data that is stored in a similar manner to how it is stored in the main memory.

- The binary files can be created only from within a program and their contents can only be read by a program.
- More secure as they are not easily readable.
- They are generally stored with **.bin** file extension.

- C supports a number of functions that have the ability to perform basic file operations, which include:

  1. Create a file
  2. Open a file
  3. Close a file
  4. Read from a file
  5. Write data into a file

- File operation functions in C:
  - ⇒ fopen() - Creates a new file for use Opens a new existing file for use
  - ⇒ fclose() - Closes a file which has been opened for use
  - ⇒ getc() - Reads a character from a file
  - ⇒ putc() - Writes a character to a file
  - ⇒ fprintf() - Writes a set of data values to a file
  - ⇒ fscanf() - Reads a set of data values from a file
  - ⇒ getw() - Reads a integer from a file
  - ⇒ putw() - Writes an integer to the file
  - ⇒ fseek() - Sets the position to a desired point in the file
  - ⇒ ftell() - Gives the current position in the file
  - ⇒ rewind() - Sets the position to the begining of the file

❖ FILE POINTER:

- FILE is a (hidden)structure that needs to be created for opening a file.
- A FILE ptr that points to this structure & is used to access the file.

FILE *fptr;

Opening a File:

```
FILE *fptr;
fptr = fopen("filename" , mode);
```

Closing a File:

fclose(fptr);

A file must be closed as soon as all operations on it have been completed. This would close the file associated with the file pointer.

❖ FILE MODE:

r – Opens a file in read mode and sets pointer to the first character in the file

w – Opens a file in write mode. It returns null if file could not be opened. If file exists, data are overwritten.

a – Opens a file in append mode. It returns null if file couldn't be opened.

r+ – Opens a file for read and write mode and sets pointer to the first character in the file.

w+ – opens a file for read and write mode and sets pointer to the first character in the file.

a+ – Opens a file for read and write mode and sets pointer to the first character in the file. But it can't modify existing contents.

```
FILE *p1 *p2;
p1=fopen ("Input.txt", "w");
p2=fopen ("Output.txt", "r");
….
…
fclose(p1);
fclose(p2);
```

opens two files and closes them after all operations on them are completed, once a file is closed its file pointer can be reversed on other file.

❖ **Opening a FILE :**

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

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

main()
{
```

Reading From a File

The file read operation in C can be performed using functions fscanf() or fgets(). Both the functions performed the same operations as that of scanf and gets but with an additional parameter, the file pointer. There are also other functions we can use to read from a file. Such functions are listed below:

Function	Description
fscanf()	Use formatted string and variable arguments list to take input from a file.
fgets()	Input the whole line from the file.
fgetc()	Reads a single character from the file.
fgetw()	Reads a number from a file.
fread()	Reads the specified bytes of data from a binary file.

So, it depends on you if you want to read the file line by line or character by character.

Example:

```
FILE * fptr;
fptr = fopen(“fileName.txt”, “r”);
fscanf(fptr, "%s %s %s %d", str1, str2, str3, &year);
char c = fgetc(fptr);
```

Note: *One thing to note here is that after reading a particular part of the file, the file pointer will be automatically moved to the end of the last read character.*

Write to a File

The file write operations can be performed by the functions fprintf() and fputs() with similarities to read operations. C programming also provides some other functions that can be used to write data to a file such as:

Function	Description
fprintf()	Similar to printf(), this function use formatted string and varible arguments list to print output to the file.
fputs()	Prints the whole line in the file and a newline at the end.
fputc()	Prints a single character into the file.
fputw()	Prints a number to the file.
fwrite()	This functions write the specified amount of bytes to the binary file.

Example:

```
FILE *fptr ;
fptr = fopen(“fileName.txt”, “w”);
fprintf(fptr, "%s %s %s %d", "We", "are", "in", 2012);
fputc("a", fptr);
```

```c
#include<stdio.h>
main()
{
    FILE *fptr;
    char data[50]="This is a demo file for file management. ";

    fptr=fopen("DemoFile.txt","w");

    if( fptr==NULL )
    {
        printf("\n\n DemoFile.txt is faled to Open..");
    }

    else
    {
        printf("\n\n File is now opened...");

        if(fptr!=EOF)
        {
            fputs(data, fptr);
            fputs("\n",fptr);
        }

        fclose(fptr);
        printf("\n\n Data successfully written in file..");
        printf("\n\n File is now closed..");
    }
}
```

```c
#include<stdio.h>

main()
{
    FILE *fptr;
    char data[40];

    fptr=fopen("DemoFile.txt","r");

    if(fptr==NULL)
    {
        printf("\n\n DemoFile is failed to open... ");
    }

    else
    {
        while(fgets(data,40,fptr)!=NULL)
        {
            printf("%s",data);
        }

        fclose(fptr);
        printf("\n\n Data from DemoFile successfully read...");
    }

}
```

```c
#include<stdio.h>
main()
{
    FILE *fptr;
    char data[50]="This is a demo file for file management. ";

    fptr=fopen("DemoFile.txt","a");

    if( fptr==NULL )
    {
        printf("\n\n DemoFile.txt is faled to Open..");
    }

    else
    {
        printf("\n\n File is now opened...");

        if(fptr!=EOF)
        {
            fputs(data, fptr);
            fputs("\n",fptr);
        }

        fclose(fptr);
        printf("\n\n Data successfully written in file..");
        printf("\n\n File is now closed..");
    }
}
```