Course Name: O Level (2nd Sem B1, B2 and B3 Batch) **Subject:** C Language
Topic: File Handling (Continued..) **Date:** 11-June-2020

File Handling

- A collection of data or information that are stored on a computer known as file
- A file is a collection of bytes stored on a secondary storage device.
- There are four different types of file
  - Data Files
  - Text Files
  - Program Files
  - Directory Files
- Different types of file store different types of information
- A file has a beginning and an end.
- We need a marker to mark the current position of the file from the beginning (in terms of bytes) while reading and write operation, takes place on a file.
- Initially the marker is at the beginning of the file. We can move the marker to any other position in the file.
- The new current position can be specified as an offset from the beginning the file.

What is File Stream?

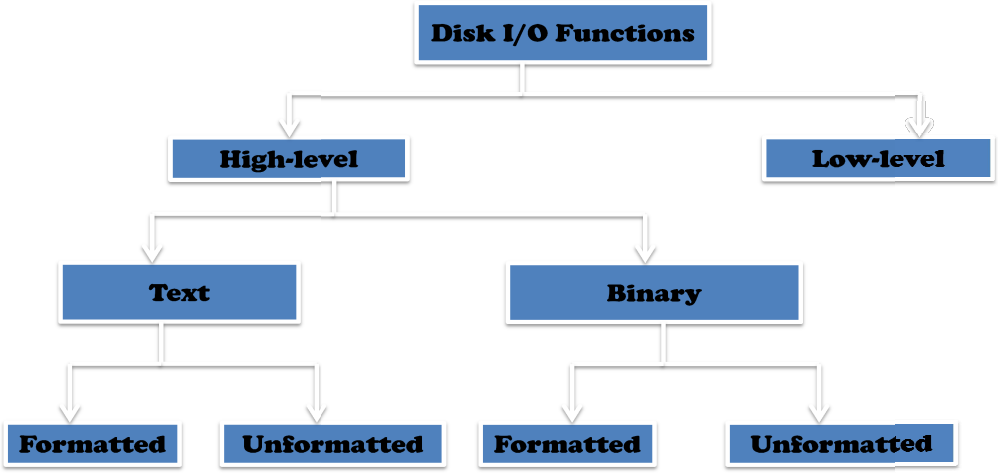
- A stream refers to the flow of data (in bytes) from one place to another (from program to file or vice-versa).
- There are two types of streams
  - **Text Stream**
    - It consists of sequence of characters
    - Each line of characters in the stream may be terminated by a newline character.
    - Text streams are used for textual data, which has a consistent appearance from one environment to another or from one machine to another
  - **Binary Stream**
    - It is a series of bytes.
    - Binary streams are primarily used for non-textual data, which is required to keep exact contents of the file.

Types of File

- **ASCII Text files**
  - A text file can be a stream of characters that a computer can process sequentially.
  - It is processed only in forward direction.
  - It is opened for one kind of operation (reading, writing, or appending) at any given time.
  - We can read only one character at a time from a text file.

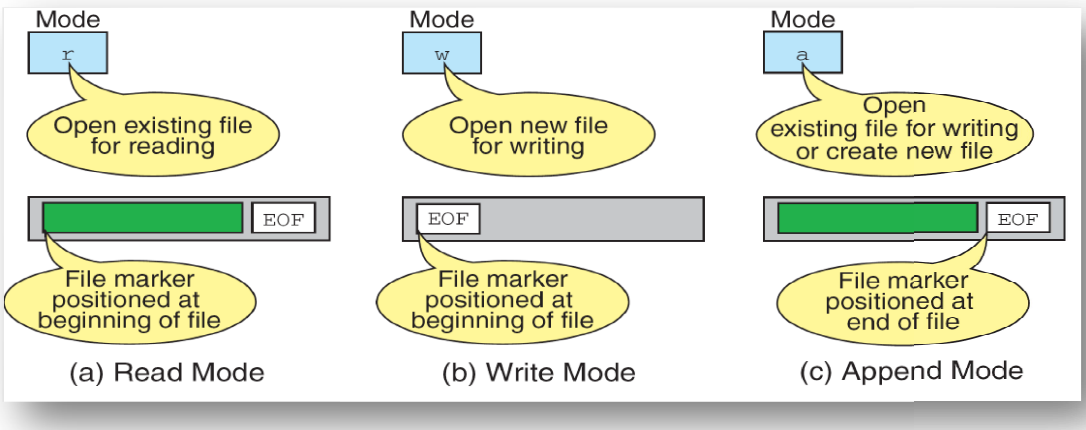
- **Binary File**
  - A binary file is a file consisting of collection of bytes.
  - A binary file is also referred to as a character stream.

Classification of file I/O functions



Opening Modes of File

Mode	Meaning
r	▪ Open a text file for reading only. If the file doesn't exist, it returns null.
w	▪ Opens a file for writing only. ▪ If file exists, than all the contents of that file are destroyed and new fresh blank file is copied on the disk and memory with same name ▪ If file doesn't exists, a new blank file is created and opened for writing. ▪ Returns NULL if it is unable to open the file
a	▪ Appends to the existing text file ▪ Adds data at the end of the file. ▪ If file doesn't exists then a new file is created. ▪ Returns NULL if it is unable to open the file.
rb	▪ Open a binary file for reading
wb	▪ Open a binary file for reading
ab	▪ Append to a binary file
r+	▪ Open a text file for read/write
w+	▪ Opens the existing text file or Creates a text file for read/write
a+	▪ Append or create a text file for read/write



Opening a File

- The general format for declaring and opening a file is:
  FILE *fp;
  fp=fopen("filename",mode");

- Here, the first statement declares the variable *fp* as a "pointer to the data type FILE".
- The second statement opens the file named *filename* with the purpose *mode* and the beginning address of the buffer area allocated for the file is stored by file pointer *fp*.

  Note: *Any no. of files can be opened and used at a time.*

Opening a File

- The closing a file ensures that all outstanding information associated with the file is flushed out from the buffers and all links to the file are broken.
- In cases where there is a limit to the no. of files that can be kept open simultaneously, closing of unwanted files help in opening the required ones.
- Another instance where we have to close a file is when we want to reopen the same file in different mode.
- ***fclose()*** returns ***0*** if the file is closed successfully.
- The file is closed using library function ***fclose()*** as:
  fclose(fp);

- The ***fcloseall()*** closes all the files opened previously.

Reading a File

- To read contents from an existing file, we need to open that file in read mode that means **"r"** mode
- Algorithm to read data from a file:
  - Open the file in read mode
  - Read data from the file
  - Write the data into an output device
  - Repeat steps 3 and 4 until the end of file occurs
  - Stop procedure.

Some high-level I/O functions

Function Name	Operation
fopen()	Create a new file for use Opens an existing file for use
fclose()	Closes file which was opened for us
fgetc()	Read a character from a file
fputc()	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
fseek()	Sets the position to a desired point in the file
ftell()	Gives the current position in the file (in terms of bytes from the start)
rewind()	Sets the position to the beginning of the file

Example 1: Display contents of a file on screen.

```
# include<stdio.h>
# include<conio.h>
void main( )
{
FILE *fp ;
char ch ;
clrscr();
fp = fopen ( "PR1.C", "r" ) ;
while (1)
{
ch = fgetc ( fp ) ;
if ( ch == EOF )
break ;
printf ( "%c", ch ) ;
}
fclose ( fp ) ;
}
```

On execution of above program it displays the contents of the file 'PR1.C' on the screen.

Let us now understand how it does the same.

Before we can read (or write) information from (to) a file on a disk we must open the file. To open the file we have called the function **fopen( )**. It would open a file "PR1.C" in 'read' mode, which tells the C compiler that we would be reading the contents of the file.

Note that "r" is a string and not a character; hence the double quotes and not single quotes. In fact **fopen( )** performs three important tasks when you open the file in "r" mode:

- Firstly it searches on the disk the file to be opened.
- Then it loads the file from the disk into a place in memory called buffer.
- It sets up a character pointer that points to the first character of the buffer.

fopen( ) in a structure called **FILE**. **fopen( )** returns the address of this structure, which we have collected in the structure pointer called **fp**. We have declared **fp** as

FILE *fp ;

Once the file has been opened for reading using **fopen( )**, as we have seen, the file's contents are brought into buffer (partly or wholly) and a pointer is set up that points to the first character in the buffer. This pointer is one of the elements of the structure to which **fp** is pointing.

To read the file's contents from memory there exists a function called **fgetc( )**. This has been used in our program as,

ch = fgetc (fp);

fgetc( ) reads the character from the current pointer position, advances the pointer position so that it now points to the next character, and returns the character that is read, which we collected in the variable **ch**.

Note that once the file has been opened, we no longer refer to the file by its name, but through the file pointer **fp**. We have used the function **fgetc( )** within an indefinite **while** loop.

There has to be a way to break out of this **while**. When shall we break out... the moment we reach the end of file. But what is end of file? A special character, whose ASCII value is 26, signifies end of file. This character is inserted beyond the last character in the file, when it is created.

While reading from the file, when **fgetc( )** encounters this special character, instead of returning the character that it has read, it returns the macro EOF. The EOF macro has been defined in the file "stdio.h". In place of the function **fgetc( )** we could have as well used the macro **getc( )** with the same effect.

In the above program we go on reading each character from the file till end of file is not met. As each character is read we display it on the screen. Once out of the loop, we close the file.