5.8 WORKING WITH TEXT FILES AND BINARY FILES.

Files are collection of records (or) it is a place on hard disk, where data is stored permanently.

Types of Files:

There are two types of files in C language which are as follows −

- Text file
- Binary File

Text File

- It contains alphabets and numbers which are easily understood by human beings.
- An error in a text file can be eliminated when seen.
- In text file, the text and characters will store one char per byte.
- For example, the integer value 4567 will occupy 2 bytes in memory, but, it will occupy 5 bytes in text file.
- The data format is usually line-oriented. Here, each line is a separate command.

Binary file

- It contains 1's and 0's, which are easily understood by computers.
- The error in a binary file corrupts the file and is not easy to detect.
- In binary file, the integer value 1245 will occupy 2 bytes in memory and in file.
- A binary file always needs a matching software to read or write it.
- For example, an MP3 file can be produced by a sound recorder or audio editor, and it can be played in a music player.
- MP3 file will not play in an image viewer or a database software.

Files are classified into following

- **Sequential files** − Here, data is stored and retained in a sequential manner.
- **Random access Files** − Here, data is stored and retrieved in a random wa

Differentiators between these two file types:

Aspect	Binary File	Text File
Data Representation	Stores data in binary format (***1s*** and ***0s***).	Stores data as ASCII characters, making it human-readable.
Use Cases	It is ideal for storing custom data like images, audio, and mixed data types.	It is suited for storing user-friendly, plain text data. Commonly used for documents, configuration files, etc.
Memory Consumption	Occupies memory based on the number of bytes in binary format.	Uses more memory due to character-based storage (1 byte per character).
Newline Handling	No automatic conversion of newline characters.	Converts newline characters to carriage return-line feed combinations.
Accessibility	Requires custom applications or software for data interpretation.	Can be viewed and edited using simple text editors.
End of File Marker	Typically tracks the end of the file based on the number of characters present.	Uses a unique ***ASCII value (26)*** as an end-of-file marker.
Data Security	Data is encrypted, making it secure but challenging to understand.	Data is less secure, but errors can be easily identified and corrected.

Error Handling	A single error can corrupt the entire file, challenging to rectify.	Errors are easier to spot and fix due to human-readable format.

C File Operations

C file operations refer to the different possible operations that we can perform on a file in C such as:

1. Creating a new file – **fopen() with attributes as “a” or “a+” or “w” or “w+”**
2. Opening an existing file – **fopen()**
3. Reading from file – **fscanf() or fgets()**
4. Writing to a file – **fprintf() or fputs()**
5. Moving to a specific location in a file – **fseek(), rewind()**
6. Closing a file – **fclose()**

Functions for C File Operations

File operation	Declaration & Description
fopen() - To open a file	Declaration: FILE *fopen (const char *filename, const char *mode) fopen() function is used to open a file to perform operations such as reading, writing etc. In a C program, we declare a file pointer and use fopen() as below. fopen() function creates a new file if the mentioned file name does not exist. FILE *fp; fp=fopen ("filename", "'mode"); Where, fp – file pointer to the data type "FILE". filename – the actual file name with full path of the file. mode – refers to the operation that will be performed on the file. Example: r, w, a, r+, w+ and a+. Please refer below the description for these mode of operations.
fclose() - To close a file	Declaration: int fclose(FILE *fp); fclose() function closes the file that is being pointed by file pointer fp. In a C program, we close a file as below. fclose (fp);
fgets() - To read a file	Declaration: char *fgets(char *string, int n, FILE *fp) fgets function is used to read a file line by line. In a C program, we use fgets function as below. fgets (buffer, size, fp); where, buffer – buffer to put the data in. size – size of the buffer fp – file pointer
fprintf() - To write into a file	Declaration: int fprintf(FILE *fp, const char *format, ...);fprintf() function writes string into a file pointed by fp. In a C program, we write string into a file as below. fprintf (fp, "some data"); or fprintf (fp, "text %d", variable_name);

Read and Write in a Binary File

Till now, we have only discussed text file operations. The operations on a binary file are similar to text file operations with little difference.

Opening a Binary File

To open a file in binary mode, we use the rb, rb+, ab, ab+, wb, and wb+ access mode in the fopen() function. We also use the .bin file extension in the binary filename.

Example

```
fptr = fopen("filename.bin", "rb");
```

Write to a Binary File

We use fwrite() function to write data to a binary file. The data is written to the binary file in the from of bits (0's and 1's).

Syntax of fwrite()

size_t**fwrite**(const void **ptr*, size_t*size*, size_t*nmemb*, FILE **file_pointer*);

Parameters:

- **ptr:** pointer to the block of memory to be written.
- **size:** size of each element to be written (in bytes).
- **nmemb:** number of elements.
- **file_pointer**: FILE pointer to the output file stream.

Return Value:

- Number of objects written.

Example:

Program to write to a Binary file using fwrite() C

```
// C program to write to a Binary file using fwrite()
#include<stdio.h>
#include<stdlib.h>
structthreeNum{
intn1,n2,n3;
};
intmain()
{
intn;
// Structure variable declared here.
StructthreeNum num;
FILE*fptr;
if((fptr=fopen("C:\\program.bin","wb"))==NULL){
printf("Error! opening file");
// If file pointer will return NULL
// Program will exit.
exit(1);
```

```
}
intflag=0;
// else it will return a pointer to the file.
for(n=1;n<5;++n){
num.n1=n;
num.n2=5*n;
num.n3=5*n+1;
flag=fwrite(&num,sizeof(structthreeNum),1,
fptr);
}
// checking if the data is written
if(!flag){
printf("Write Operation Failure");
}
else{
printf("Write Operation Successful");
}
fclose(fptr);
return0;
}
```

Output

Write Operation Successful