1. What is a file?

- A file represents a **sequence of bytes**, regardless of it being a **text file** or a **binary file**, on the disk where a group of related data is stored.
- File is created for **permanent storage of data**. It is a **ready-made structure**.

Declaration: **FILE *fp;**

- use a structure **pointer of file type** to declare a file.

2. In what include file is the data type FILE defined?

- **<stdio.h>**

3. Syntax and Semantics of the ff operations:

fopen()

- Creates a new file or open an existing file.
- It also returns a file pointer to it.
- Always check return value to make sure you don't get NULL.
- You can use the fopen() function to create a new file or to open an existing file. This call will initialize an object of the type FILE, which contains all the information necessary to control the stream.

Syntax: FILE *fopen (const char * filename, const char * mode);
Ex: FILE *ptr = fopen ("sample/hello.txt", "operation");

"operation" can be the ff:

r	Opens an existing file for reading purposes.
w	Opens a file for writing. If file doesn't exist, a new file is created. It removes the content of the file if it exists.
a	Opens a file for appending. If file doesn't exist, a new file is created.
r+	Opens file for both reading and writing.
w+	Opens file for both reading and writing. It truncates file to zero length if it exists, otherwise creates a new file.
a+	Opens file for both reading and writing. If it does not exist, then it creates a new file. Reading mode starts at the beginning. Writing can only be appended.
wx	Opens a file for writing. If it exists, it returns null.

If handling binary files, use: "rb", "wb", "ab", "rb+", "r+b", "wb+", "w+b", "ab+", "a+b"

==========

fclose()

- Closes the file. Close the file after using in a particular mode.
- The **fclose()** function returns zero on success, or **EOF (-1 in value)** if there is an error in closing the file. This function actually flushes any data still pending in the buffer to the file, closes the file, and releases any memory used for the file. The EOF is a constant defined in the header file **stdio.h**.

Syntax: int fclose (FILE *fp);

==========

fwrite()

- Function to write in a binary file.
- Use this when writing files in a binary file.

Syntax: size_t fwrite(const void *ptr, size_t size, size_t number_of_elements, FILE *stream);

Ex: fwrite (&myStructure, sizeof(structure), 1, fp);

Where:

ptr - This is pointer to array of elements to be written

size - This is the size in bytes of each element to be written

number_of_elements - This is the number of elements, each one with a size of size bytes

stream - This is the pointer to a FILE object that specifies an output stream

---

Ex: fwrite(<buffer>, <size>, <qty>, <file pointer>);

- Writes <qty> units of size <size> to the file pointed to (<file pointer>) by reading them from a buffer (usually an array) pointed to by <buffer>.
- Note: the operation of the file pointer passed in as a parameter must be “w” for write or “a” for append, or you will suffer an error.

==========

fread()

- Function to read from a binary file.
- Use this when reading from a binary file.

Syntax: size_t fread(void *ptr, size_t size, size_t number_of_elements, FILE *stream);

Ex: fread(&myStructure, sizeof(structure), 1, fp);

Where:

ptr - This is the pointer to a block of memory with a minimum size of size*number_of_elements bytes

size - This is the size in bytes of each element to be read

number_of_elements - This is the number of elements, each one with a size of size bytes

stream - This is the pointer to a FILE object that specifies an input stream

---

Ex: fread(<buffer>, <size>, <qty>, <file pointer>);

- reads <qty> units of size <size> from the file pointed to and stores them in memory in a buffer (usually an array) pointed to by <buffer>
- note: the operation of the file pointer passed in as a parameter must be "r" for read, or you will suffer an error.

==========

fseek()

- used to move the reading control to different positions.
- 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.
- Returns 0 if successful. Non-zero if unsuccessful.
- Use this when locating specific position in a very big file.

Syntax: int fseek(FILE * stream, long int offset, int whence);

Ex: fseek(fptr, -sizeof(struct threeNum), SEEK_END);

Where:

stream – this is the pointer to a FILE object that identifies the stream.

offset – this is the number of bytes to offset from whence.

whence – this is the position from where offset is added.

Where **whence** can be the ff:

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

==========

ftell()

- tells the byte location of current position of cursor in file pointer.
- Returns the current file position of the given stream.
- If error occurs, returns -1L.
- Use this when wanting to know current position of cursor in file.

Syntax: long int ftell(FILE *stream);

Ex: ftell(fp);

- In a file “Hello World, and Hello Everyone!”, and fseek(fp, 0, SEEK_END);, the function ftell(fp) will return the length of the file.

==========

rewind()

- it moves the control or file position to beginning of the file.
- Sets the file position indicator for the stream pointed to by stream to the beginning of the file. It is equivalent to (void)fseek(stream, 0L, SEEK_SET) except that the error indicator of the stream is also cleared.
- Use this when wanting to traverse file from the beginning, AGAIN.

Syntax: void rewind(FILE *stream);

Ex: rewind(fp);

- when fp has travelled the file all the way to EOF, using rewind will let fp point to the beginning of the file, AGAIN.