FILE OPERATIONS

The various operations involved in using files in C are

1. Declaring the file pointer variable
2. Opening the file
3. Reading data from file
4. Writing data to the file
5. Closing the file

Declaring the file pointer variable

A file pointer variable has to be declared in order to access a particular file.

Syntax:

```
FILE *file_pointer_name;
```

Example:

```
FILE *fp;
```

Here, fp is the file pointer variable.

Opening the file

A file must be opened before read/write operation. The function fopen() is used to open a file and associate it with a stream. It returns a pointer to FILE structure if the file is opened successfully, otherwise a NULL is returned.

Syntax:

```
FILE *fopen(const char *file_name, const char *mode);
```

Here, file_name - Every file has a file name. In fopen(), the file name is specified along with the path of the file in the disk.

Example:

E:\\CSE\\Student.DAT (To represent a backslash in C, another backslash is used.)

mode - Mode represents the type of processing that can be done with the file. The following table shows the different modes of processing the file.

Mode	Description
r	Open a text file for reading
w	Open a text file for writing
a	Append to a text file.
rb	Open a binary file for reading
wb	Open a binary file for writing
ab	Append to a binary file.
r+	Open an existing text file for read/write
w+	Create a new text file for read/write
a+	Append a text file for read/write
r+b/rb+	Open an existing binary file for read/write
w+b/wb+	Create a new binary file for read/write
a+b/ab+	Append a binary file for read/write

File Accessing Modes

Reading Data From File

Data can be read from files using the following functions.

a) fscanf()
b) fgets()
c) fgetc()
d) fread()

Function	Description
fscanf	Used to read formatted data from the stream. **Syntax:** int fscanf(FILE *stream, const char *format, ...);
fgets()	Used to read a line from the stream. **Syntax**: char *fgets(char *str, int n, FILE *stream);
fgetc()	Used to read the next character from the stream. **Syntax**: int fgetc(FILE *stream);

fread()	Used to read number of elements specified by num of size specified by size from the stream. **Syntax:** int fread(void *str, size_t size, size_t num, FILE *stream);

Functions to Read Data from File

Writing Data to File

The functions used to write data to files are,

- fprintf()
- fputs()
- fputc()
- fwrite()

Function	Description
fprintf()	Used to write formatted data to the stream. **Syntax:** int fprintf(FILE *stream, const char *format, ...);
fputs()	Used to write a line to the stream. **Syntax:** int fputs(const char *str, FILE *stream);
fputc()	Used to write a character to the stream. **Syntax:** int fputc(int char, FILE *stream);
fwrite()	Used to write number of elements specified by num of size specified by size to the stream. **Syntax:** fwrite(const void *str, size_t size, size_t num, FILE *stream);

Functions to Write Data to File

Closing a File

The file (both text and binary) should be closed after reading/writing. The function fclose() is used to close an already opened file. It disconnects the file pointer and flushes out all

the buffers associated with the file.

Syntax:

int fclose(FILE *fp);

Here fp is the file pointer of the file that has to be closed. The function returns zero for successful completion and a non-zero value otherwise.

The function fcloseall() closes all the streams except the standard streams(stdin, stdout, stderr) and flushes out the stream buffers.

Syntax:

int fcloseall(void);

Detecting the End-Of-File(EOF)

EOF is a symbolic constant defined in the library function stdio.h with a value -1. There are two ways to detect an EOF.

- Compare the characters that has been read with the value of EOF (ie.,-1).
- Use the standard library function feof() defined in stdio.h. It returns one if EOF has been reached and zero otherwise.

Syntax:

int feof(FILE *fp);

Program: Program To Open, Write And Close A File

```
# include <stdio.h>
# include <string.h>
int main( )
{
        FILE *fp ;
        char data[50];
        printf( "Opening the file sample.c in write mode" ) ;
        fp = fopen("sample.c", "w") ;                    // opening an existing file
        if ( fp == NULL )
        {
            printf( "Could not open file sample.c" ) ;
           return 1;
        }
```

```
printf( "\n Enter some text from keyboard" );    // getting input from user
while ( strlen ( gets( data ) ) > 0 )
{                                                 // writing in the file
      fputs(data, fp) ;
      fputs("\n", fp) ;
}
printf("Closing the file sample.c") ;
fclose(fp) ;                                      // closing the file
return 0;       }
```

Output:

Opening the file sample.c in write mode

Enter some text from keyboard

Hai, How are you?

Closing the file sample.c

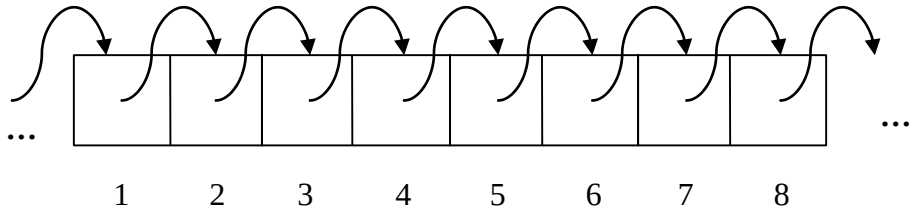
TYPES OF FILE PROCESSING

There are two types of processing based on how a file is accessed. They are,

- Sequential Access
- Random Access

Sequential Access

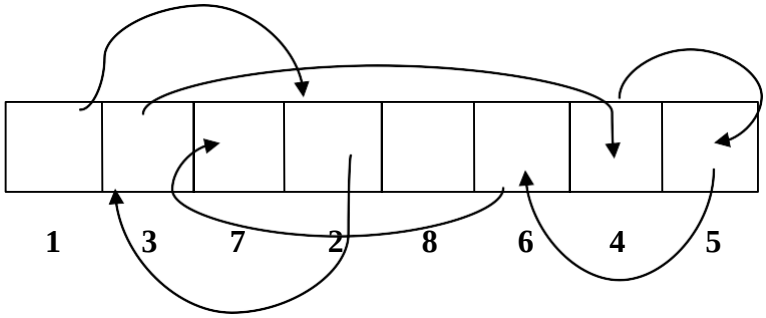
In Sequential Access, the program processes the data in a sequential manner. ie, one by one. In order to read the last record, all the records before that record need to be read.



Sequential Access

Random Access

In Random Access, the program accesses the file at the point at which the data should be read or written. The last record in the file can be read directly.



Random Access