File Handling in C

1. Introduction to File handling

A file can be thought of as a collection of bytes stored on a secondary storage device, which is usually called a disk.

The collection of bytes could form a text document with characters, words, lines, paragraphs and pages, or a database with its fields and records, or a graphical image made of pixels, or some other data.

2. What is file handling

File handling in C enables us to create, update, read, and delete the files stored on the local file system through our C program.

File handling in C refers to the task of storing data in the form of input or output produced by running C programs in data files, namely, a text file or a binary file for future reference and analysis.

3. Why We need Files

We need files during programming due to following reasons

- ✓ When a program is terminated, the entire data is lost. Storing in a file will preserve your data even if the program terminates.
- ✓ If you need to enter a large number of data, it will take a lot of time to enter them all. However, if you have a file containing all the data, you can easily access the contents of the file using a few commands in C.
- ✓ You can easily move your data from one computer to another without any changes.

The meaning attached to a particular file dependent on what we do with the file. It is determined by the data structures and operations used by a program to process the file.

Irrespective of the data they contain, or the methods used to process them, all files have certain common properties.

1. Every file has a name.
2. The file must be opened before we start processing it. Conceptually, until a file is opened no operation can be performed on the file. (When it is opened, we may access it at its beginning or end. To prevent accidental misuse, at the time of opening the file we must tell the system what we intend to do with the file – read, write, or append.)
3. It must be closed when it is done with. (When we are finished using the file, we must close it. If the file is not closed, the operating system

cannot finish updating its own housekeeping records used for file management, and data in the file may be lost.)

4. The third one is about the actual operations on the file. In between the opening and closing of a file, we can write into, or read from, or append to it.

4. Types of Files

Essentially there are two kinds of files that programmers deal with – text files and binary files.

4.1 Text File

- ✓ Text files are used to store character data.
- ✓ A text file consists of a stream of characters that can be processed sequentially. This sequential processing is possible only in the forward direction. Because of this restriction, a text file is usually opened for only one kind of operation (read, write or append) at any given time.
- ✓ Similarly, since text files only process characters, they can only read or write data one character at a time. There are functions that read or write a string or a line of text, but these functions also essentially process the data one character at a time.

4.2 Binary Files

- ✓ Instead of storing data in plain text, they store it in the binary form (0's and 1's).
- ✓ They can hold a higher amount of data, are not readable easily, and provides better security than text files.
- ✓ The major difference is in the way the file is processed.
    - Binary files can be processed sequentially or using direct access, depending on the needs of the application. In C, processing a file with a direct access involves moving the current file position to an appropriate place in the file before reading or writing data.
    - Since it is possible to have access that is not sequential, binary files are generally opened for read and write operations simultaneously. but are rarely found in applications that process text files.

The operations performed on binary files are similar to text files. In fact, the same functions are used for reading and writing operations on files in C, for both text and binary files. We make the distinction between text and binary only at the time of opening a file, when we have to indicate the type of file being processed.

5. File Operations

In C, you can perform four major operations on files, either text or binary:

- ✓ Creating a new file or Opening an existing file
- ✓ Closing a file
- ✓ Reading from (Input) the file

- ✓ writing (output) information to a file

6. working with file

When working with files, you need to declare a pointer to structure of type FILE. This declaration is needed for communication between the file and the program.

```
FILE *fp;
```

Here, `*fp` is the FILE pointer (`FILE *fp`), which will hold the reference to the opened (or created) file.

7. Opening in file or creating a file

Before we can write to a file, we must open it. What this really means is that we must tell the system that we want to write to a file and what the filename is.

The `fopen()` function is used to create a new file or to open an existing file.

The syntax for opening a file in standard I/O

```
FILE *fopen (const char *filename, const char *mode)
```

or

```
FILE *ptr = fopen("fileopen","mode");
```

fp – file pointer to the structure data type “FILE”.
filename – the actual file name with full path of the file. The filename is any valid filename that the operating system can understand. It is enclosed in double quotes.
mode – refers to the operation that will be performed on the file. Example: r, w, a, r+, w+ and a+.

Mode can be of following types,

mode	Description
r	opens a text file in reading mode
w	opens or create a text file in writing mode.
a	opens a text file in append mode
r+	opens a text file in both reading and writing mode
w+	opens a text file in both reading and writing mode
a+	opens a text file in both reading and writing mode
rb	opens a binary file in reading mode
wb	opens or create a binary file in writing mode
ab	opens a binary file in append mode
rb+	opens a binary file in both reading and writing mode
wb+	opens a binary file in both reading and writing mode
ab+	opens a binary file in both reading and writing mode

Using the r indicates that the file is assumed to be a text file. Opening a file for reading requires that the file already exists. If it does not exist, the file pointer variable will be set to NULL by the fopen( ) function.

When a file is opened for writing, it will be created if it does not already exist and it will be reset if it does, resulting in the deletion of any data already there. Using the w indicates that the file is assumed to be a text file.

When a file is opened for appending, it will be created if it does not already exist and it will be initially empty. If a file exists with that name, the data input point will be positioned at the end of the present data so that any new data will be added to any data that already exists in the file. Using the a indicates that the file is assumed to be a text file.

8. Closing a File

One should always close a file whenever the operations on file are over. It means the contents and links to the file are terminated. This prevents accidental damage to the file.
The `fclose()` function is used to close an already opened file.

General Syntax :

```
int fclose( FILE *fp);
```

- The `fclose` function takes a file pointer as an argument and closes the file.
- It returns zero on success, or `EOF` if there is an error in closing the file.
- This EOF is a constant defined in the header file `stdio.h`.
- After closing the file, the same file pointer can also be us with other files.
- In 'C' programming, files are automatically close when the program is terminated. Closing a file manually by writing `fclose` function is a good programming practice.

Example:

```
FILE *fp;
fp  = fopen ("data.txt", "r");
fclose (fp);
```

9. Writing to a File

In C, when you write to a file, newline characters '\n' must be explicitly added.

The `stdio.h` library offers the necessary functions to write to a file:

- `fputc(char, file_pointer)`: It writes a character to the file pointed to by file_pointer.
- `fputs(str, file_pointer)`: It writes a string to the file pointed to by file_pointer.
- `fprintf(file_pointer, str, variable_lists)`: It prints a string to the file pointed to by file_pointer. The string can optionally include format specifiers and a list of variables variable_lists.

The program below shows how to perform writing to a file:

9.1 fputc() Function:

- It writes a character to the file pointed to by file_pointer.
- Syntax
  `fputc(char, file_pointer):`

```
#include <stdio.h>
int main() {
        int i;
        FILE * fptr;
        char fn[50];
        char str[] = "Sir Padampat Singhania University\n";
        fptr = fopen("test.txt", "w"); // "w" defines "writing mode"
        for (i = 0; str[i] != '\n'; i++) {
            /* write to file using fputc() function */
            fputc(str[i], fptr);
        }
        fclose(fptr);
        return 0;
    }
```

The above program writes a single character into the test.txt file until it reaches the next line symbol "\n".

- In the above program, we have created and opened a file called fputc_test.txt in a write mode and declare our string which will be written into the file.
- We do a character by character write operation using for loop and put each character in our file until the "\n" character is encountered then the file is closed using the `fclose()` function.

9.2 fputs() Function:

- It writes a string to the file pointed to by file_pointer.
- Syntax

```
fputs(str, file_pointer):
```

```
#include <stdio.h>
int main() {
        FILE * fp;
        fp = fopen("fputs_test.txt", "w+");
        fputs("Sir Padampat Singhania University,", fp);
        fputs("Department of Computer Science and Engineering\n", fp);
        fputs("CS-1001( Strucutred Programming Approach\n", fp);
        fclose(fp);
        return (0);
    }
```

In the above program, we have created and opened a file called fputs_test.txt in a write mode.
After we do a write operation using `fputs()` function by writing three different strings Then the file is closed using the `fclose` function.

9.3 fprintf()

- It prints a string to the file pointed to by file_pointer. The string can optionally include format specifiers and a list of variables variable_lists.
- Syntax

```
fprintf(file_pointer, str, variable_lists):
```

- Example

```
void main()
{
        char empname[20];
        int empno,salary,n=3,i;
        FILE *sfptr;
        ptr=fopen("file3","w");

        printf("\n Employee details \n");
        for(i=0;i<3;i++)
        {
        scanf("%d %s %d", &empno,empname,&salary);
        fprintf(ptr,"%d %s %d\n",empno,empname,salary);
        }
        fclose(ptr);
```

```
}
```

Above Program is reading three values(empno, empname, and salary from the keyboard using standard input function `scanf( )` and writing these 3 values in the file file3(pointed by file pointer `*ptr.`)

10. Reading data from a File

There are three different functions dedicated to reading data from a file

- `fgetc(file_pointer)`: It returns the next character from the file pointed to by the file pointer. When the end of the file has been reached, the EOF is sent back.
- `fgets(buffer, n, file_pointer)`: It reads n-1 characters from the file and stores the string in a buffer in which the NULL character '\0' is appended as the last character.
- `fscanf(file_pointer,conversion_specifiers, variable_adresses)`: It is used to parse and analyze data. It reads characters from the file and assigns the input to a list of variable pointers variable_adresses using conversion specifiers. Keep in mind that as with scanf, fscanf stops reading a string when space or newline is encountered.

10.1 fgetc() function

The `fgetc()` function reads a character from the input file referenced by `fp`. The return value is the character read, or in case of any error, it returns `EOF`.

Syntax

```
int fgetc( FILE * fp );
```

Example

```
#include <stdio.h>
int main()
{
     /* Pointer to the file */
     FILE *fp1;
     /* Character variable to read the content of file */
     char c;

     /* Opening a file in r mode*/
     fp1= fopen ("C:\\myfiles\\newfile.txt", "r");

     /* Infinite loop –I have used break to come out of the loop*/
     while(1)
     {
        c = fgetc(fp1);
        if(c==EOF)
            break;
        else
```

```
            printf("%c", c);
     }
     fclose(fp1);
     return 0;
}
```

10.2 fgets() function

The functions `fgets()` reads up to n-1 characters from the input stream referenced by `fp`. It copies the read string into the buffer `buf`, appending a null character to terminate the string.

Syntax

```
char *fgets(char *buf, int rec_len, FILE *fpr)
```

buf: Array of characters to store strings.
rec_len: Length of the input record.
fpr: Pointer to the input file.
Example

```
#include <stdio.h>
int main()
{
    FILE *fpr;
    /*Char array to store string */
    char str[100];
    /*Opening the file in "r" mode*/
    fpr = fopen("C:\\mynewtextfile.txt", "r");

    /*Error handling for file open*/
    if (fpr == NULL)
    {
       puts("Issue in opening the input file");
    }

    /*Loop for reading the file till end*/
    while(1)
    {
       if(fgets(str, 10, fpr) ==NULL)
            break;
       else
            printf("%s", str);
    }
    /*Closing the input file after reading*/
    fclose(fpr);
    return 0;
}
```

10.3 fscanf() function

- It is used to parse and analyze data.
- It reads characters from the file and assigns the input to a list of variable pointers variable_adresses using conversion specifiers.
- Keep in mind that as with `scanf()`, `fscanf()` stops reading a string when space or newline is encountered.

Syntax

```
int fscanf(FILE *fp, const char *format, ...)
```

Example

```
#include<stdio.h>
#include<conio.h>
struct emp
{
   char name[10];
   int age;
};

main()
{
   struct emp e;
   FILE *p,*q;
   p = fopen("one.txt", "a");
   q = fopen("one.txt", "r");

   printf("Enter Name and Age");
   scanf("%s %d", e.name, &e.age);
   fprintf(p,{"%s %d", e.name, e.age);
   fclose(p);
   do
   {
      fscanf(q,"%s %d", e.name, e.age);
      printf("%s %d", e.name, e.age);
   }
   while(!eof(q));
   getch();
}
```