File Handling in C

In programming, we may require some specific input data to be generated several numbers of times. Sometimes, it is not enough to only display the data on the console. The data to be displayed may be very large, and only a limited amount of data can be displayed on the console, and since the memory is volatile, it is impossible to recover the programmatically generated data again and again. However, if we need to do so, we may store it onto the local file system which is volatile and can be accessed every time.

Why files are needed?

- 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 have 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.

Types of Files

When dealing with files, there are two types of files you should know about:

1. Text files
2. Binary files

1. Text files

- Text files are the normal **.txt** files. You can easily create text files using any simple text editors such as Notepad.
- When you open those files, you'll see all the contents within the file as plain text. You can easily edit or delete the contents.
- They take minimum effort to maintain, are easily readable, and provide the least security and takes bigger storage space.

2. Binary files

- Binary files are mostly the **.bin** files in your computer.
- 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.

File handling in C enables us to create, update, read, and delete the files stored on the local file system through our C program. The following operations can be performed on a file.

- Creation of the new file
- Opening an existing file
- Reading from the file
- Writing to the file
- Moving to a specific location in a file
- Closing a file

Functions for file handling

There are many functions in the C library to open, read, write, search and close the file. A list of file functions are given below:

No.	Function	Description
1	fopen()	opens new or existing file
2	fprintf()	write data into the file
3	fscanf()	reads data from the file
4	fputc()	writes a character into the file
5	fgetc()	reads a character from file
6	fclose()	closes the file
7	fseek()	sets the file pointer to given position
8	fputw()	writes an integer to file
9	fgetw()	reads an integer from file
10	ftell()	returns current position
11	rewind()	sets the file pointer to the beginning of the file

Opening File: fopen()

We must open a file before it can be read, write, or update. The fopen() function is used with the required access modes to open a file. The syntax of the fopen() is given below.

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

Example

```
fopen("fileopen","mode");
fopen("E:\\cprogram\\newprogram.txt","w");

fopen("E:\\cprogram\\oldprogram.bin","rb");
```

The fopen() function accepts two parameters:

- The file name (string). If the file is stored at some specific location, then we must mention the path at which the file is stored. For example, a file name can be like **"c:\\some_folder\\some_file.txt"**.
- The mode in which the file is to be opened. It is a string.

We can use one of the following modes in the fopen() function.

Mode	Description
r	opens a text file in read mode
w	opens a text file in write mode
a	opens a text file in append mode
r+	opens a text file in read and write mode
w+	opens a text file in read and write mode
a+	opens a text file in read and write mode
rb	opens a binary file in read mode
wb	opens a binary file in write mode
ab	opens a binary file in append mode
rb+	opens a binary file in read and write mode
wb+	opens a binary file in read and write mode
ab+	opens a binary file in read and write mode

The fopen function works in the following way.

- Firstly, it searches the file to be opened.
- Then, it loads the file from the disk and place it into the buffer. The buffer is used to provide efficiency for the read operations.
- It sets up a character pointer which points to the first character of the file.

As given above, if you want to perform operations on a binary file, then you have to append 'b' at the last. For example, instead of "w", you have to use "wb", instead of "a+" you have to use "ab+". For performing the operations on the file, a special pointer called File pointer is used which is declared as

```
FILE *filePointer;
So, the file can be opened as
filePointer = fopen("fileName.txt", "w")
```

Consider the following example which opens a file in write mode.

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

OUTPUT

```
#include;
void main( )
{
FILE *fp; // file pointer
char ch;
fp = fopen("file_handle.c","r");
while ( 1 )
{
ch = fgetc ( fp ); //Each character of the file is read and stored in the
character file.
if ( ch == EOF )
break;
printf("%c",ch);
}
fclose (fp );
}
```

Closing a File

The file (both text and binary) should be closed after reading/writing. Closing a file is performed using the `fclose()` function. The syntax of fclose() function is given below:

int fclose(FILE *fp);

Example

```
fclose(fptr);
```

Here, `fptr` is a file pointer associated with the file to be closed.

C fprintf() and fscanf()

Writing File : fprintf() function

The fprintf() function is used to write set of characters into file. It sends formatted output to a stream.

Syntax:

int fprintf(FILE *stream, const char *format [, argument, ...])

Example:

```
#include <stdio.h>
main(){
   FILE *fp;
   fp = fopen("file.txt", "w");//opening file
   fprintf(fp, "Hello file by fprintf...\n");//writing data into file
   fclose(fp);//closing file
}
```

Reading File : fscanf() function

The fscanf() function is used to read set of characters from file. It reads a word from the file and returns EOF at the end of file.

Syntax:

int fscanf(FILE *stream, const char *format [, argument, ...])

Example:

```
#include <stdio.h>
main(){
   FILE *fp;
   char buff[255];//creating char array to store data of file
   fp = fopen("file.txt", "r");
   while(fscanf(fp, "%s", buff)!=EOF){
   printf("%s ", buff );
   }
   fclose(fp);
}
```

Output:

```
Hello file by fprintf...
```

C File Example: Storing employee information

Let's see a file handling example to store employee information as entered by user from console. We are going to store id, name and salary of the employee.

```
#include <stdio.h>
void main()
{
    FILE *fptr;
    int id;
    char name[30];
    float salary;
    fptr = fopen("emp.txt", "w+");/*  open for writing */
    if (fptr == NULL)
    {
        printf("File does not exists \n");
        return;
    }
    printf("Enter the id\n");
    scanf("%d", &id);
    fprintf(fptr, "Id= %d\n", id);
    printf("Enter the name \n");
    scanf("%s", name);
    fprintf(fptr, "Name= %s\n", name);
    printf("Enter the salary\n");
    scanf("%f", &salary);
    fprintf(fptr, "Salary= %.2f\n", salary);
    fclose(fptr);
}
```

Output:

```
Enter the id
1
Enter the name
sonoo
Enter the salary
120000
```

Now open file from current directory. For windows operating system, go to TC\bin directory, you will see emp.txt file. It will have following information.

emp.txt

```
Id= 1
Name= sonoo
Salary= 120000
```

C fputc() and fgetc()

Writing File : fputc() function

The fputc() function is used to write a single character into file. It outputs a character to a stream.

Syntax:

```
int fputc(int c, FILE *stream)
```

Example:

```
#include <stdio.h>
main(){
  FILE *fp;
  fp = fopen("file1.txt", "w");//opening file
  fputc('a',fp);//writing single character into file
  fclose(fp);//closing file
}
```

file1.txt

```
a
```

Reading File : fgetc() function

The fgetc() function returns a single character from the file. It gets a character from the stream. It returns EOF at the end of file.

Syntax:

```
int fgetc(FILE *stream)
```

Example:

```
#include<stdio.h>
#include<conio.h>
void main(){
FILE *fp;
char c;
clrscr();
fp=fopen("myfile.txt","r");

while((c=fgetc(fp))!=EOF){
printf("%c",c);
}
fclose(fp);
getch();
}
```

myfile.txt

```
this is simple text message
```

C fputs() and fgets()

The fputs() and fgets() in C programming are used to write and read string from stream. Let's see examples of writing and reading file using fgets() and fgets() functions.

Writing File : fputs() function

The fputs() function writes a line of characters into file. It outputs string to a stream.

Syntax:

```
int fputs(const char *s, FILE *stream)
```

Example:

```
#include<stdio.h>
#include<conio.h>
void main(){
FILE *fp;
clrscr();

fp=fopen("myfile2.txt","w");
fputs("hello c programming",fp);

fclose(fp);
getch();
}
```

myfile2.txt

```
hello c programming
```

Reading File : fgets() function

The fgets() function reads a line of characters from file. It gets string from a stream.

Syntax:

```
char* fgets(char *s, int n, FILE *stream)
```

Example:

```
#include<stdio.h>
#include<conio.h>
void main(){
FILE *fp;
char text[300];
clrscr();

fp=fopen("myfile2.txt","r");
printf("%s",fgets(text,200,fp));

fclose(fp);
getch();
}
```

Output:

```
hello c programming
```

Reading and writing to a binary file

Functions `fread()` and `fwrite()` are used for reading from and writing to a file on the disk respectively in case of binary files.

Writing to a binary file

To write into a binary file, you need to use the `fwrite()` function. The functions take four arguments:

1. address of data to be written in the disk
2. size of data to be written in the disk
3. number of such type of data
4. pointer to the file where you want to write.

```
fwrite(addressData, sizeData, numbersData, pointerToFile);
```

Example

```
#include <stdio.h>
#include <stdlib.h>

struct threeNum
{
   int n1, n2, n3;
};

int main()
{
   int n;
   struct threeNum num;
   FILE *fptr;

   if ((fptr = fopen("C:\\program.bin","wb")) == NULL){
       printf("Error! opening file");
```

```
        // Program exits if the file pointer returns NULL.
        exit(1);
    }

    for(n = 1; n < 5; ++n)
    {
       num.n1 = n;
       num.n2 = 5*n;
       num.n3 = 5*n + 1;
       fwrite(&num, sizeof(struct threeNum), 1, fptr);
    }
    fclose(fptr);

    return 0;
}
```

In this program, we create a new file `program.bin` in the C drive.

We declare a structure `threeNum` with three numbers - *n1, n2 and n3*, and define it in the main function as num.

Now, inside the for loop, we store the value into the file using `fwrite()`.

The first parameter takes the address of *num* and the second parameter takes the size of the structure `threeNum`.

Since we're only inserting one instance of *num*, the third parameter is `1`. And, the last parameter `*fptr` points to the file we're storing the data.

Finally, we close the file.

Reading from a binary file

Function `fread()` also take 4 arguments similar to the `fwrite()` function as above.

```
fread(addressData, sizeData, numbersData, pointerToFile);
```

Example

```
#include <stdio.h>
#include <stdlib.h>

struct threeNum
{
   int n1, n2, n3;
};

int main()
{
   int n;
   struct threeNum num;
   FILE *fptr;

   if ((fptr = fopen("C:\\program.bin","rb")) == NULL){
       printf("Error! opening file");

       // Program exits if the file pointer returns NULL.
       exit(1);
   }

   for(n = 1; n < 5; ++n)
   {
      fread(&num, sizeof(struct threeNum), 1, fptr);
      printf("n1: %d\tn2: %d\tn3: %d", num.n1, num.n2, num.n3);
   }
   fclose(fptr);

   return 0;
}
```

In this program, you read the same file `program.bin` and loop through the records one by one.

In simple terms, you read one `threeNum` record of `threeNum` size from the file pointed by **fptr* into the structure *num*.

We 'll get the same records you inserted in previous **example**.

C fseek() function

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.

Syntax of fseek()

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

The first parameter stream is the pointer to the file. The second parameter is the position of the record to be found, and the third parameter specifies the location where the offset starts.

Different whence in fseek()

Whence	**Meaning**
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.

Example:

```
#include <stdio.h>
void main(){
   FILE *fp;

   fp = fopen("myfile.txt","w+");
   fputs("This is VSSUT", fp);

   fseek( fp, 7, SEEK_SET );
   fputs("Ritik", fp);
   fclose(fp);
}
```

myfile.txt

```
This is Ritik
```

Example

```
#include <stdio.h>
#include <stdlib.h>

struct threeNum
{
   int n1, n2, n3;
};

int main()
{
   int n;
   struct threeNum num;
   FILE *fptr;

   if ((fptr = fopen("C:\\program.bin","rb")) == NULL){
       printf("Error! opening file");

       // Program exits if the file pointer returns NULL.
       exit(1);
   }
   // Moves the cursor to the end of the file
   fseek(fptr, -sizeof(struct threeNum), SEEK_END);

   for(n = 1; n < 5; ++n)
   {
      fread(&num, sizeof(struct threeNum), 1, fptr);
      printf("n1: %d\tn2: %d\tn3: %d\n", num.n1, num.n2, num.n3);
      fseek(fptr, -2*sizeof(struct threeNum), SEEK_CUR);
   }
   fclose(fptr);

   return 0;
}
```

This program will start reading the records from the file program.bin in the reverse order (last to first) and prints it.

C rewind() function

The rewind() function sets the file pointer at the beginning of the stream. It is useful if you have to use stream many times.

Syntax:

```
void rewind(FILE *stream)
```

Example:

File: file.txt

```
this is a simple text
```

File: rewind.c

```
#include<stdio.h>
#include<conio.h>
void main(){
FILE *fp;
char c;
clrscr();
fp=fopen("file.txt","r");

while((c=fgetc(fp))!=EOF){
printf("%c",c);
}

rewind(fp);//moves the file pointer at beginning of the file
while((c=fgetc(fp))!=EOF){
printf("%c",c);
}

fclose(fp);
getch();
}
```

Output:

```
this is a simple textthis is a simple text
```

As you can see, rewind() function moves the file pointer at beginning of the file that is why "this is simple text" is printed 2 times. If you don't call rewind() function, "this is simple text" will be printed only once.

C ftell() function

The ftell() function returns the current file position of the specified stream. We can use ftell() function to get the total size of a file after moving file pointer at the end of file. We can use SEEK_END constant to move the file pointer at the end of file.

Syntax:

```
long int ftell(FILE *stream)
```

Example:

File: ftell.c

```
#include <stdio.h>
#include <conio.h>
void main (){
  FILE *fp;
  int length;
  clrscr();
  fp = fopen("file.txt", "r");
  fseek(fp, 0, SEEK_END);

  length = ftell(fp);

  fclose(fp);
  printf("Size of file: %d bytes", length);
  getch();
}
```

Output:

Size of file: 21 bytes