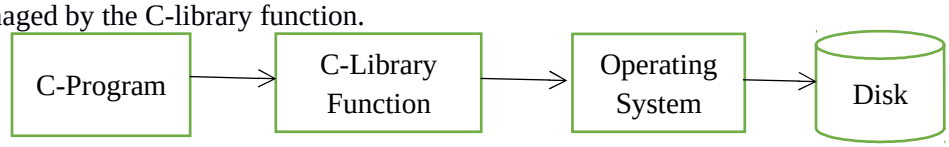
CHAPTER – 9

File Handling

1. Introduction

File is a data structure which helps us to store our data permanently on the secondary storage devices such as hard disk, floppy disk, or CD etc. This is important because the computer memory is volatile in nature and also large data cannot display on the screen at a time.

All the data stored on disk is in binary form and the data structure may varies from one operating system to another but this is all managed by the C-library function.



The files accessed through the library functions are called Stream Oriented files and the files accessed with system calls are known as System Oriented files.

2. File Operations_

C supports various operations on file such as: creating a new file, opening an existing file, reading data from a file, writing data to a file, moving to a specific location in a file (seeking) and closing a file. The general logic performed on file can be listed as:

- Define a local 'pointer' of type FILE (called file pointer)
- Open the file and associate it with the file pointer via fopen()
- Perform the I/O operations using file I/O functions (ex. fscanf() and fprintf())
- Disconnect the file from the task using fclose()

For example:

```
FILE *fp;
fp = fopen("name", "mode");
fscanf(fp, "format string", variable list);
fprintf(fp, "format string", variable list);
fclose(fp );
```

3. File Handling function on C_

Function	Operations
fopen ()	After creating new file, it opens an existing file for use. The fopen() returns the file pointer position for successful open and returns NULL if the file does not open or the file does not exist.
fclose ()	Close a file which that has been opened for use. The fclose() returns zero for successful close and returns EOF (end of file) when error is encountered in closing a file. By default, all the files opened are closed when the program is terminated. It is good to close all the files opened with fopen(), because files can be reopened only if they are closed.
putc ()	Write a character to a file. **Syntax:** putc (char_variable, file_pointer) For example: putc (c, fp);
getc ()	Read a character from a file. **Syntax**: char_variable = getc (file_pointer) For example: c = getc (fp);
putw ()	Write an integer to a file. **Syntax**: = putw (int_variable, file_pointer) For example: putw (2, fp);
getw ()	Read an integer from a file. **Syntax**: int_variable = getw (file_pointer)

	For example: 2 = getw (fp);
fprintf ()	Write a set of data value to a file.
	Syntax: fprintf(file_pointer,"data_format", variable_list);
	For example: fprintf (fp, "%s %d", name, roll_no);
fscanf ()	Read a set of different data type value from a file.
	Syntax: fscanf(file_pointer,"data_format", &variable_list);
	For example: fscanf (fp, "%s %d", name, &roll_no);

Note:
The file pointer moves one by one variable position in every operations of input-output. The input function will return a EOF (end of file) marked when the end of file has been reached. For EOF, ctrl + z (^z) is used.

3.1 Creating a file

Before performing any task on file, we must create a new file.
Syntax:
FILE *file_pointer_name; //FILE must be in capital letter
Example:
FILE *fp;

3.2 Opening a file

Before we can write a file to disk, or read it, we must open it. Opening a file establishes an understanding between our program and the operating system. This may be the opening mode of file, operation on file etc.
Syntax:
file_pointer_name = fopen("file_name", "Operation_mode");
Example:
FILE *fptr; //creating a file
fptr=fopen("abc.txt","w"); //opening a file

File opening modes

r	Open an existing file for reading only
w	Opens a new file for writing only. If a file with specified filename currently exists, it will be destroyed and a new file is created in its place.
a	Opens an existing file for appending (i.e. adding new information at the end of file). A new file will be created if the file with the specified filename does not exist.
r+(update mode+)	Open an existing file for both reading and writing.
w+	Open a new file for both reading and writing. If a file with specified filename currently exists, it will be destroyed and a new file is created in its place.
a+	Open an existing file for both reading and writing. A new file will be created if the file with the specified filename does not exist.

3.3 Closing a file

When all the input-output operations on a file are completed, we need to close it. Closing a file means all the links to the file is broken. This is carried out with the statement:
fclose(fptr); //where, fptr is the file pointer of FILE

4. Random File Access in C

Random file access means that you can take the file pointer to any part of the file for reading or writing. In general, with small files, we access the files sequentially. In sequential access, we access the file record by record or character by character. This approach is appropriate if the file size is small, but what if the file has 10000's of records? In random file access, if a file has 10000 records and if you want to go to the 9999th record, you can directly point the file pointer to this record using the random file access technique. This can be done using the functions mentioned below:

- **fseek()**
  The fseek() function helps us send a file pointer to a specified location. The syntax of fseek() is as follows:

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

The function accepts three parameters; the file pointer, an integer offset (which is the number of bytes to be shifted from the position mentioned in the third parameter), and the third parameter which specifies the position from where the offset is added. Along with this, there are three macros used in fseek(). These are SEEK_SET (beginning of the file), SEEK_CUR (current position in the file), SEEK_END (end of the file).

- **ftell()**
  The ftell() function is used to find out the exact position of the file pointer with respect to the beginning. This function accepts the file pointer as the parameter.

  long ftell(FILE *pointer);

- **rewind()**
  This function also accepts the file pointer as a parameter and sets the file pointer to the beginning of the file again. The syntax is as follows:

  void rewind(FILE *pointer);

Example

```
#include <stdio.h>
int main () {
  FILE *fp;
  int c;
  fp = fopen("file.txt","w+");
  // we are using fseek to shift the file pointer to the 7th position
  fseek( fp, 7, SEEK_SET );
 //Now we overwrite C programming in the 7th position
  fputs(" C Programming", fp);
 //now we print the current position of the file pointer using ftell
  printf("The current position of the file pointer is: %ld\n", ftell(fp));
//we take the file pointer to the beginning of the file
  rewind(fp);
 //now we verify if rewind() worked using ftell
  printf("The current position of the file pointer is: %ld\n", ftell(fp));
  while(1)
{
   c = fgetc(fp);
   if( feof(fp) ) {
    break;
   }
   printf("%c", c);
  }
  fclose(fp);
  return(0);
}
```

5. File Error

Each FILE* that you use to read and write data from and to a file contains flags that the system sets when certain events occur. If you get an error, it sets the error flag; if you reach the end of the file during a read, it sets the EOF flag. Pretty simple really.
The functions feof() and ferror() give you a simple way to test these flags: they'll return non-zero (true) if they're set. Once the flags are set for a particular stream, they stay that way until you call clearerr() to clear them.

- **Return Value**
  feof() and ferror() return non-zero (true) if the file has reached EOF or there has been an error, respectively.

```
// read binary data, checking for eof or error
int main(void)
{
    int a;
    FILE *fp;

    fp = fopen("binaryints.dat", "rb");

    // read single ints at a time, stopping on EOF or error:

    while(fread(&a, sizeof(int), 1, fp), !feof(fp) && !ferror(fp)) {
        printf("I read %d\n", a);
    }

    if (feof(fp))
        printf("End of file was reached.\n");

    if (ferror(fp))
        printf("An error occurred.\n");

    fclose(fp);

    return 0;
}
```

Examples:
Example1: Program to write character data to a text file (character file)

```
 #include  <stdio.h>
#include  <conio.h>
void  main()
{
FILE  *fp;
char  c;
fp=fopen("sample.txt","w");
while  ((c=getchar())!=EOF)
{
putc(c,fp);
}
fclose(fp);
getch();
clrscr();
}
```

Example2: Program to read a character data from a text file

```
#include  <stdio.h>
#include  <conio.h>
void  main()
{
FILE  *fp;
char  c;
fp=fopen("sample.txt","r");
while  ((c=getc(fp))!=EOF)
{
printf("%c",c);
```

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

Example3: Program to write string data to a file

```
#include  <stdio.h>
#include  <conio.h>
void  main()
{
FILE  *fp;
char  c[50];
fp=fopen("sample.txt","w");
gets(c);
fputs(c,fp);
fclose(fp);
getch();
clrscr();
}
```

Example4: Program to read string data from a file

```
#include  <stdio.h>
#include  <conio.h>
void  main()
{
FILE  *fp;
char  c[50];
fp=fopen("sample.txt","r");
fgets(c,10,fp);
printf("%s",c);
fclose(fp);
getch();
clrscr();
}
```

Example5: WAP to open a new file, read name, address and telephone number of 10 employees from the user and write to a file. (Exam 2013 Fall)

```
#include<stdio.h>
#include<conio.h>
void main()
{
        FILE *f;
        char name[20],address[20];
        long telno;
        int i;
        f=fopen("employee.txt","w");
        for(i=0;i<10;i++)
        {
                scanf("%s%s%ld",name,address,&telno);
                fprintf(f,"%s%s%ld\n",name,address,telno);
        }
        fclose(f);
        getch();
}
```

Example6: WAP to read information of 10 students in n subjects. Store in a file and print the result.(Exam 2008 Fall)

Name	C	CPP	Math	Eco	Total
Ram	80	85	90	75	330

… … … … … …
… … … … … …

```
#include<stdio.h>
#include<conio.h>
void main()
{
int i,c,cpp,math,eco,total;
char name[15];
FILE *f;
f=fopen("Student.txt","w");
for(i=0;i<2;i++)
{
scanf("%s%d%d%d%d",name,&c,&cpp,&math,&eco);
total=c+cpp+math+eco;
fprintf(f,"%s\t%d\t%d\t%d\t%d\t%d\n",name,c,cpp,math,eco,total);
printf("\n%s\t%d\t%d\t%d\t%d\t%d",name,c,cpp,math,eco,total);
}
fclose(f);
getch();
clrscr();
}
```

Note: this can be solved by another way. By first opening in write mode and then read mode. But question has not stated it compulsory.

Example7: Create a structure for the following data:
ID, name, address, salary, and date of joining(dd/mm/yy)
And write program to input 100 employee, save n emp.data file of employee who have the address “Pokhara”. (Exam 2009 Fall)

```
#include<stdio.h>
#include<conio.h>
struct employee
{
        int ID;
        char name[20];
        char address[20];
        float salary;
        int dd;
        int mm;
        int yy;
};
void main()
{
FILE *f;
int i;
struct employee e;
f=fopen("emp.dat","w")   ;
for(i=0;i<100;i++)
{
        scanf("%d%s%s%f%d%d%d",&e.ID,e.name,e.address,&e.salary,&e.dd,&e.mm,&e.yy);
        if(e.address=="pokhara")
        {
                fprintf(f,"%d\t%s\t%s\t%f\t%d\t%d\t%d\
n",e.ID,e.name,e.address,e.salary,e.dd,e.mm,e.yy);
        }
}
fclose(f);
getch();
clrscr();
}
```

Example8: WAP to create a structure for customer in a shop of the following data:
CID, cname, address, telno, balance.
Read 100 customers from a file customer.dat and display the information of the customers who has the balance greater than Rs.1000.(Exam 2009 Fall)

```
#include<stdio.h>
#include<conio.h>
struct cust
{
        int CID;
        char cname[20];
char address[15];
long telno;
float balance;
};
void main()
{
FILE *f;
int i;
struct cust e;
f=fopen("customer.dat","r")      ;
for(i=0;i<100;i++)
{
        fscanf(f,"%d%s%s%ld%f ",&e.CID,e.cname,e.address,&e.telno,&e.balance);
        if(e.balance>1000)
        {
                printf("%d%s%s%ld%f,e.CID,e.cname,e.address,e.telno,e.balance);
        }
}
fclose(f);
getch();
clrscr();
}
```