Unit 9
File Handling

Introduction: File

A file is a place on the disk where a group of related data is stored. File handling is a process of creating a new file, append records in file and read records from file i.e. File handling in C provides the facility to store and manipulate inputted or processed information in a file for further use.

The program that accepts the input data from the keyboard at the time of execution and writes output to the VDU (visual display unit), such type of Input / Output is called console I/O. for the purpose we use ***printf(), scanf(), getch(), getche(), getchar(), gets(), puts()***, etc functions. It is fine for small amount of data. It has two main problems:

- It becomes very inconvenient and time consuming to handle the large volume of data through the terminal.
- The entire data is lost when either the program is terminated or the computer system is turned off.

There are two types of data files:

- High level
- Low level

High level files are also sub-divided into two categories:

- Text files
- Binary files

Text files

A text file is a human-readable sequence of character and the words they form that can be encoded into computer readable format such as ASCII. A text file contains only textual characters like alphabets, digits, and special symbols with no special formatting such as underlining or displaying character in bold face or different fonts. There is no graphical information, sound or video files. A text a file is also knows as ASCII file can be read by any word processor. Text file stores information in consecutive characters. These characters can be interpreted as individual data items or as component of strings or numbers.
A good example of a text file is any C program, say *simple_interest.c*

Binary files

A binary file is merely a collection of bytes. This collection might be a complied version of C program (say *simple_interest.exe).*or music data stored in wave file or a picture stored in graphic file. It contains more than plain text e.g. sound, image, graphics etc. a binary file made up of machine readable symbols that represents 1's and 0's. These files organize data in to block containing contagious bytes information.

Opening modes:

rb, rb+, wb, wb+, ab and ab+

Difference between text and binary file:

a) Handling of new line

In text mode, a new line character is converted into carriage return-line feed combination before writing it to the disk, while reading back from the disk, the carriage-return line-feed combination is converted back into a new line. If a file is opened in binary mode, these conversions are not required.

b) Representation of End of file (EOF)

In text mode, a character having ASCII value 26 is inserted at the end of file to mark EOF. The read function would return EOF signal, if this character is encountered while reading the file. There is no such special character at the end of binary file to mark the end of file. The binary mode file keeps track of the end of the file from the number of character present in the directory entry of the file.

These two modes are not compatible. So any file written in text mode must be read back in the text mode and in binary mode must be read in binary mode.

c) Storage of numbers:

In text mode, numbers are stored as string of characters. The number 32235 occupies 2 bytes in memory but when it is written to disk using function *fprintf()*, it occupies 5 bytes. Similarly, 12345 occupy 4 bytes in memory but 11 bytes in disk. I.e. in text mode, there is insufficient space on the disk, if large amount of numerical data is to be stored.

In binary mode, there is sufficient space on the disk to store large amount of data. Each number would occupy the same number of bytes on both memory and disk.

File Operations:

There are different operations that can be carried out on a file. These are:

a) Creation of a new file
b) Opening an existing file
c) Reading from a file
d) Writing to a file
e) Moving to a specific location in a file
f) Closing a file

Opening a file

To work with file using any file handling library functions, we have to declare the file using structure **FILE**. The structure **FILE** is defined in <*stdio.h*> header file. It can be declared as:

Syntax: FILE *file_pointer:
Example: FILE *fp;

A structure named FILE is defined in the file stdio.h that contains all the information about file like:

- Name of file
- Status
- Buffer size
- current position
- end of file status

Before we can read or write information from or to a file on a disk we must open the file. To open a file **fopen()** function is used.

Syntax: `file pointer = fopen("file_name", "mode");`
Example: `fp = fopen("record.txt", "r");`

The **fopen ( )** performs three important tasks.

- Firstly it searches on the disk the file to be opened
- Then it loads the file from the disk into a place in memory called buffer
- It sets up a character pointer that points to the first character of the buffer

Errors in fopen()

If an error occurs in opening a file, then fopen() returns NULL.

```
FILE *p;
p= fopen("abc.txt", "r");
if(p==NULL)
   {
   printf("Error in opening file");
   exit(1);
   }
```

Errors may occur due to following reasons

- If we try to open a file in read mode and If the file doesn't exists or we do not have read permission on that file.
- If we try to create a file but there is no space on disk or we don't have write permissions.
- If we try to create a file that already exists and we don't have permission to delete that file.
- Operating system limits the number of files that can be opened at a time and we are trying to open more files than that number.

Reading from a file

Once the file has been opened by using **fopen(),** we can read or write from or into the file. To read the file's contents from the memory, the function **fgetc()** is used.

Syntax: `variable = fgetc(file_pointer);`
Example: `ch = fgetc (fp);`

The **fgetc()** reads the character from the current pointer position, advances the pointer position so that it now points to the next character, and return the character that is read, which can be collected in the variable.

Closing a file

Once we have finished reading from the file, we need to close it. For this, we can use a function: **fclose ().**

Syntax: `fclose (file_pointer);`
Example: `fclose (fp);`

Once we close the file, we cannot read from it unless we reopen the file. When we close the file using **fclose()**, three operations would be performed:

a) The characters in the buffer would be written to the file on the disk.
b) At the end of file a character with ASCII value 26 (EOF) would get written.
c) The buffer would be eliminated from memory.

File Opening Modes:

The list of all possible file opening modes is given below.

Modes	Meanings
"r"	Open a text file for reading Syntax: filepointer = fopen("filename", "r");
"w"	Creates a text file for writing but if the text file exists, overwrites the contents. Syntax: filepointer = fopen("filename", "w");
"a"	Append mode either open an existing text file for writing without overwriting the previous contents or creates new text file if the file does not exist. Syntax: filepointer = fopen("filename", "a");
"r+"	Reading existing contents, writing new contents, modifying existing contents of the file i.e. for both reading and writing Syntax: filepointer = fopen("filename", "r+");
"w+"	Creates a text file for both reading and writing but if text file exist it just overwrites the contents. Syntax: filepointer = fopen("filename", "w+");
"a+"	Opens a text file for reading and writing preserving previous contents. Creates new file if the specified file does not exist. Syntax: filepointer = fopen("filename", "a+");
rb	Opens a binary file for reading. Syntax: filepointer = fopen("filename", "rb");

wb	Creates a new binary file for writing but if the binary file exists, overwrites the contents. Syntax: filepointer = fopen("filename", "wb");
ab	Append mode either open an existing binary file for writing without overwriting the previous contents or creates new binary file if the file does not exist. Syntax: filepointer = fopen("filename", "ab");
rb+	Open binary file for both reading and writing. Syntax: filepointer = fopen("filename", "rb+");
wb+	Creates a binary file for both reading and writing but if binary file exist it just overwrites the contents. Syntax: filepointer = fopen("filename", "wb+");
ab+	Opens a binary file for reading and writing preserving previous contents. Creates new binary file if the specified file does not exist. Syntax: filepointer = fopen("filename", "ab+");

Formatted Input / Output

The C language provides a set of library functions to perform input and output (I/O) operations. Those functions can read or write any type of data to files. There are two functions: fscanf() and fprintf() to read and write from an to the file.

fprintf()

It is used to write set of characters into file. It sends formatted output to a stream.

Syntax: `fprintf(file_pointer, "control strings", arguments);`

The following program illustrates the working mechanism of *fprintf()*

```
int i= 12;
float x = 2.356;
char ch = 's';
FILE *fp;
fp=fopen("out.txt","w");
fprintf(fp, "%d %f %c", i, x, ch);
```

fscanf()

It 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: `fscanf(file_pointer, "control strings", arguments);`

Write a program to store employee information such as employee id, name, and salary entered by user from console into the file.

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

struct employee
      {
      int emp_id;
      char name[40];
      float salary;
      };

void main()
{
FILE *fp;
struct employee e;
```

```
clrscr();
fp=fopen("Employee.txt","w+");
if(fp==NULL)
      {
      printf("\n File doesn't exist");
      return;
      }

printf("\n Input Employee ID:- ");
scanf("%d", &e.emp_id);
fprintf(fp,"Employee ID = %d\n", e.emp_id);
fflush(stdin);
printf("\n Input Name:- ");
gets(e.name);
fprintf(fp, "Name = %s\n", e.name);
printf("\n Input Salary:- ");
scanf("%f", &e.salary);
fprintf(fp,"Salary = %10.2f",e.salary);
fclose(fp);
getch();
}
```

fputc()

It is used to write a single character into file. It outputs a character to a stream.
Syntax: int `fputc(int char, FILE *stream);`

Where

- Char – It is the character to be written, it is passes as its int promotion.
- Steam – It is the pointer to a FILE object that identifies the stream where the character is to be written

The working of *fputc()* is demonstrated in following program.

```
#include<stdio.h>
#include<conio.h>
void main()
{
FILE *fp;
int ch;
fp = fopen("myfile.txt","w+");
for(ch = 65; ch<=90;ch++)
      fputc(ch, fp);
fclose(fp);
}
```

Output:
ABCDEFGHIJKLMNOPQRSTUVWXYZ

fgetc()

It is used to return a single character from the file. It gets a character from the stream. It return EOF at the end of the file.
Syntax: `fgetc(file_pointer);`

The working of *fgetc()* is demonstrated in following program.

```
#include<stdio.h>
#include<conio.h>
void main()
```

```
{
FILE *fp;
char c;
clrscr();
fp=fopen("employee.txt","r");
while((c=fgetc(fp))!=EOF)
      printf("%c",c);
fclose(fp);
getch();
}
```

fseek()

It is used to set the file pointer to the specified offset. It is used to write data into file at desired location. This function sets the file position indicator for the stream pointed to by stream or we can say it seeks a specified place within a file and modify it. This function accepts three arguments: file stream, offset and whence. The first argument: FILE stream pointer returned by the *fopen()* function, *offset* tells the amount of bytes to seek and the third argument *whence* tells from where they seek of *offset* number of bytes is to be done. The available values for whence are:

SEEK_SET	Seeks from beginning of file
SEEK_CUR	Seeks from current position
SEEK_END	Seeks from end of file

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

The working of *fseek()* is demonstrated in following program

```
#include<stdio.h>
#include<conio.h>
#include<stdlib.h>
void main()
{
FILE *fp;
clrscr();
fp=fopen("myfile.txt","w+");
fputs("This is Example of fseek function ",fp);
fseek(fp,7, SEEK_SET);
fputs(" I Love Programming ",fp);
fclose(fp);
getch();
}
```

In above program, Initially program creates the file and writes ***This is Example of fseek function*** but later we had reset the write pointer at 7th position from the beginning and used puts() statement which over-write the file and provides following output.

Output:
This is I Love Programming

rewind()

It sets the file pointer at the beginning of the stream. It is useful if we have to use steam many times.

Syntax: `rewind(file_pointer);`

The working of *rewind()* is demonstrated in following program

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

```
void main()
{
FILE *fp;
char ch;
clrscr();
fp=fopen("employee.txt","r");
while((ch=fgetc(fp))!=EOF)
      printf("%c",ch);
rewind(fp);
while((ch=fgetc(fp))!=EOF)
      printf("%c",ch);
fclose(fp);
getch();
}
```

ftell()

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

Syntax: *ftell* `(file_pointer);`

The working of *ftell()* is demonstrated in following program

```
#include<stdio.h>
#include<conio.h>
void main()
{
FILE *fp;
int length;
clrscr();
fp=fopen("employee.txt","r");
fseek(fp,0,SEEK_END);
length=ftell(fp);
fclose(fp);
printf("\n Size occupied by the file = %d bytes",length);
getch();
}
```

fread () and fwrite()

fwrite()

It is used to write binary data as well as text file to the file.

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

Remarks: fwrite appends a specified number of equal-sized data items to an output file.

- Ptr = Pointer to any object; the data written begins at ptr
- size = Length of each item of data
- n =Number of data items to be appended
- stream = file pointer

Let us consider following examples.

Example 1: Writing a variable

```
float *f = 300.85;
fwrite (&f, sizeof(f),1,fp);
```

It will write the value of variable f to the file.

Example 2: Writing an array

```
int num[5] = {100, 200, 40, 50, 890};
fwrite (num, sizeof(num),1,fp);
```

It will write the entire array into the file.

Example 3: writing some elements of array

```
int num[5] = {100, 200, 40, 50, 890};
fwrite (num, sizeof(num),2,fp);
```

It will write only the first two elements of arrays into the file.

Example 4: Writing Structures

```
struct student
    {
    char name[30];
    int roll, marks;
    }:
struct student s = {"Bharat", 1, 250};

fwrite (s, sizeof(s),1,fp);
```

It will write the contents of structure variable *s* into the file.

Example 5: Writing array of structures:

```
struct student
    {
    char name[30];
    int roll, marks;
    }:

    struct student s[3] = {
                              {"Bharat", 1, 250},
                              {"Unnav", 2, 400},
                              {"Ramesh", 3, 340}
                            };
fwrite (s, sizeof(s),1,fp);
```

It will write the whole contents of array *s* into the file.

Example 6: Writing certain contents of structure

```
struct student
   {
char name[30];
int roll, marks;
}:
Struct student s[3] = {
                            {"Bharat", 1, 250},
                            {"Unnav", 2, 400},
                            {"Ramesh", 3, 340}
                       };

fwrite (s, sizeof(s),2,fp);
```

It will write the 0^{th} and 1^{st} element of the array into the file.

fread()

It is commonly used to read binary data as well as text data too It has four arguments:

- Address of the structure to be written to the disk
- Size of structure in bytes
- Number of structure that we want to write at a time
- The pointer to the file, where we want to write.

***Syntax*:** `fread(void *ptr, size, n, FILE *stream);`

Remarks: fread reads a specified number of equal-sized data items from an input stream into a block.

- ptr = It is the reference of an array or structure where data will be stored after reading
- size = it is the total number of bytes to be read from file
- n = Number of times a record will be read.
- stream = file pointer

Let us consider following examples.

Example 1: Reading a float value from a file

```
int val;
fread (&val, sizeof(int),1,fp);
```

It will read a float value from the file and stores it in the variable *val*.

Example 2: Reading an array from the file

```
int num[10];
fread (num, sizeof(num),1,fp);
```

It will read an array of 10 integers from the file and stores in the variable *arr*.

Example 3: Reading the first five elements of an array

```
int num[10];
fread (num, sizeof(num),5,fp);
```

It will read 5 integers from the file and stores it in the variable *arr*..

Example 4: Reading the structure variable

```
struct student
    {
    char name[30];
    int roll, marks;
    }:
struct student s;

fread (&s, sizeof(s),1,fp);
```

It will read the contents of a structure variable from the file and stores it in the variable *s*.

Example 5: Reading an array of structure

```
struct student
    {
    char name[30];
    int roll, marks;
    }:

    struct student s[100]
fread (&s, sizeof(struct s),10,fp);
```

It will read first 10 elements of type struct student from the file and stores them in the variable s.

Examples related to *fread()* and *fwrite()*:

Program 1:

```
#include<stdio.h>
#include<conio.h>
struct book
      {
      char name[50];
      int page;
      };

void main()
{
FILE *fp;
int i,n;
struct book b;
clrscr();
fp=fopen("Book.dat","wb+");
if(fp==NULL)
      {
      printf("\n File can't be opened");
      exit(0);
      }
printf("\n How many books:- ");
scanf("%d", &n);
printf("\n Input following records of %d books:- ",n);
for(i=0;i<n;i++)
      {
      fflush(stdin);
      printf("\n Name of Book:- ");
      gets(b.name);
      printf("\n Number of Page:- ");
      scanf("%d", &b.page);
      fwrite(&b,sizeof(b),1,fp);
      }
fclose(fp);
clrscr();
fp=fopen("Book.dat","rb");
printf("\n Name of Book \t Number of Page");
while(fread(&b,sizeof(b),1,fp)==1)
      printf("\n %s \t %d",b.name,b.page);
fclose(fp);
getch();
}
```

Program 2:

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

struct student
      {
      char name[40];
      int roll;
      int marks;
      };
```

```
void main()
{
FILE *fp;
struct student s;
char ch;
fp=fopen("Student.dat","w");
if(fp==NULL)
      {
      printf("\n File can't be opened");
      exit(0);
      }
do
      {
      printf("\n Input Roll No.:- ");
      scanf("%d",&s.roll);
      fflush(stdin);
      printf("\n Input Name:- ");
      gets(s.name);
      printf("\n Input Total Marks:- ");
      scanf("%d",&s.marks);
      fwrite(&s,sizeof(s),1,fp);
      printf("\n Any more data (y/n):- ");
      ch=getche();
      }while(ch=='y'||ch=='Y');
printf("\n Data Written Successfully...");
fclose(fp);
getch();
}
```