# FILE HANDLING

- A file is a collection of bytes stored on a secondary storage device, which is generally a disk of some kind. The collection of bytes may be interpreted, for example, as characters, words, lines, paragraphs and pages from a textual document; fields and records belonging to a database; or pixels from a graphical image.
- The meaning attached to a particular file is determined entirely by the data structures and operations used by a program to process the file. It is conceivable (and it sometimes happens) that a graphics file will be read and displayed by a program designed to process textual data.
- The result is that no meaningful output occurs (probably) and this is to be expected. A file is simply a machine decipherable storage media where programs and data are stored for machine usage.

## Why we need file?

- 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 few commands in C. It is possible easily move your data from one computer to another without any changes.

## File Operations

- 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. Here, comes the need of file handling in C.
- 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
    - Deleting the file

## Types of Files

- There are two kinds of files in which data can be stored in two ways either in characters coded in their ASCII character set or in binary format. They are
    1. Text Files (or) ASCII file
    2. Binary Files

### Text Files (or) ASCII file

- The file that contains ASCII codes of data like digits, alphabets and symbols is called text file (or) ASCII file.

### Binary Files

- A binary file is a file that uses all 8 bits of a byte for storing the information .It is the form which can be interpreted and understood by the computer.
- The only difference between the text file and binary file is the data contain in text file can be recognized by the word processor while binary file data can’t be recognized by a word processor.

1. wb(write)

   This opens a binary file in write mode.

   SYNTAX: fp=fopen(“data.dat”,”wb”);

2. rb(read)

   This opens a binary file in read mode

   SYNTAX:fp=fopen(“data.dat”,”rb”);

3. ab(append)

   This opens a binary file in a Append mode i.e. data can be added at the end of file.

   SYNTAX: fp=fopen(“data.dat”,”ab”);

4. r+b(read+write)

   This mode opens preexisting File in read and write mode.

   SYNTAX: fp=fopen(“data.dat”,”r+b”);

5. w+b(write+read)

   This mode creates a new file for reading and writing in Binary mode.

   SYNTAX: fp=fopen(“data.dat”,”w+b”);

6. a+b(append+write)

   This mode opens a file in append mode i.e. data can be written at the end of file.

   SYNTAX: fp=fopen(“data.dat”,”a+b”);

**Opening Modes in Standard I/O**

| | | |
|---|---|---|
| r | Open for reading | If the file does not exist, fopen() returns NULL |
| rb | Open for reading in binary mode. | If the file does not exist, fopen() returns NULL. |
| w | Open for writing. | If the file exists, its contents are overwritten. If the file does not exist, it will be created. |
| wb | Open for writing in binary mode. | If the file exists, its contents are overwritten. If the file does not exist, it will be created. |

| | | |
|---|---|---|
| a | Open for append. i.e, Data is added to the end of file. | If the file does not exists, it will be created. |
| ab | Open for append in binary mode. i.e, Data is added to end of file. | If the file does not exists, it will be created. |
| r+ | Open for both reading and writing. | If the file does not exist, fopen() returns NULL. |
| rb+ | Open for both reading and writing in binary file. | If the file does not exist, fopen() returns NULL |
| w+ | Open for both reading and writing. | If the file exists, its contents are overwritten. If the file does not exist, it will be created. |
| wb+ | Open for both reading and writing in binary mode. | If the file exists, its contents are overwritten. If the file does not exist, it will be created. |
| a+ | Open for both reading and appending. | If the file does not exists, it will be created. |
| ab+ | Open for both reading and appending in binary mode. | If the file does not exists, it will be created. |

**Closing a File:** fclose(fptr);

- The file (both text and binary) should be closed after reading/writing. Closing a file is performed using library function fclose().

**Reading and writing to a text file**

- The functions fprintf() and fscanf() are used to read or write the file They are just the file versions of printf() and scanf(). The only difference is that,  fprint and fscanf expects a pointer to the structure FILE.

**Writing to a text file**

**Program 2.20: Write to a text file using fprintf()**

```
#include <stdio.h>
int main()
{
int num;
FILE *fptr;
fptr = fopen(“C:\\program.txt”,”w”);
if(fptr == NULL)
{
printf(“Error!”);
exit(1);
}
printf(“Enter num: “);
scanf(“%d”,&num);
fprintf(fptr,”%d”,num);
fclose(fptr);
return 0;
}
```

- This program takes a number from user and stores in the file program.txt. After you compile and run this program, you can see a text file program.txt created in C drive of your computer. When you open the file, you can see the integer you entered.

**Reading from a text file**

**Program 2.21: Read from a text file using fscanf()**

```
#include <stdio.h>
int main()
{
int num;
FILE *fptr;
if ((fptr = fopen(“C:\\program.txt”,”r”)) == NULL){
printf(“Error! opening file”);
// Program exits if the file pointer returns NULL.
exit(1);
}
fscanf(fptr,”%d”, &num);
printf(“Value of n=%d”, num);
fclose(fptr);
return 0;
}
```

**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 function fwrite(). The functions takes four arguments: Address of data to be written in disk, Size of data to be written in disk, number of such type of data and pointer to the file where you want to write.

fwrite(address_data,size_data,numbers_data,pointer_to_file);

**Program 2.22: Writing to a binary file using fwrite()**

```
#include <stdio.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 = 5n;
num.n3 = 5n + 1;
fwrite(&num, sizeof(struct threeNum), 1, fptr);
}
fclose(fptr);
return 0;
}
```

- We declare a structure three Num 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 three Num. 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 fwrite() function as above.

**fread(address_data,size_data,numbers_data,pointer_to_file);**

**Program 2.23: Reading from a binary file using fread()**

```
#include <stdio.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;
}
```

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

**Text Files**

- In C, all components are files, each with a different behavior based on the attached devices. To enable the I/O functions, several standard built-in functions were created and stored in libraries.
- Some of the high level file I/O functions are given in Table 2.1

**Table 2.1 High level file I/O functions**

| S.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 |

1. **fopen () :** It creates a new file for use or opens an existing file for use**.**
2. **fclose ():** It closes a file which has been opened for use.
3. **fscanf**( file pointer, format string, address of the variable)

   Example: fscanf(fptr,"%d", &num);
4. **fprintf(**console output, "format string", file pointer**);**

   Example: fprintf(stdout, "%f \n", f); /*note: stdout refers to screen */
5. **getw ():** This function returns the integer value from a given file and increment the file pointer position to the next message.

   Syntax: getw (fptr);

   Where fptr is a file pointer which takes the integer value from file.
6. **putw ():** This function is used for writing an integer value to a given file.

   Syntax: putw (value,fptr);

Where fptr is a file pointer Value is an integer value which is written to a given file.

**Example Program for getw() and putw()**

**Program 2.24: Write a program to read integer data from the user and write it into the file using putw() and read the same integer data from the file using getw() and display it on the output screen.**

```
#include<stdio.h>
#include<conio.h>
void main()
{
FILE *fp;
int n;
clrscr();
fp=fopen("c.dat", "wb+");
printf("Enter the integer data");
scanf("%d",&n);
while(n!=0)
{
        putw(n,fp);
scanf("%d",&n);
}
rewind(fp);
printf("Reading data from file");
while((n=getw(fp))!=EOF)
{
printf("%d\n",n);
}
fclose(fp);
getch();
}
```

7. **fwrite()**
   - This function is used for writing an entire block to a given file.

     Syntax: fwrite(ptr,size,nst,fptr);

     ptr is a pointer ,it points to the array of structure.

     Size is the size of the structure

     nst is the number of the structure

     fptr is a filepointer.

8. **fread()**
   - fread(ptr,size,position,fptr);similar to fwrite

9. **fflush(stdin)**;To clean the input stream

**Program 2.25: program for fwrite():**

**Write a program to read an employee details and write them into the file at a time using fwrite().**

```
#include<stdio.h>
#include<conio.h>
void main()
                {
struct emp
{
int eno;
char ename[20];
float sal;
}e;
FILE *fp;
fp=fopen(“emp.dat”, “wb”);
clrscr();
printf(“Enter employee number”);
scanf(“&d”,&e.eno);
printf(“Enter employee name”);
fflush(stdin);
scanf(“%s”,e.ename);
printf(“Enter employee salary”);
scanf(“%f”,&e.sal);
fwrite(&e,sizeof(e),1,fp);
printf(“One record stored successfully”);
getch();
}
```

**Operations for Search data in a file**

1. fseek()
2. ftell()

3. rewind()

**fseek() :** *Getting data using fseek()*

- When 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().

  Syntax of fseek()

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

  fseek(file pointer, displacement, pointer position);

- 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.
- This function is used for seeking the pointer position in the file at the specified byte.

  *Syntax:* fseek( file pointer, displacement, pointer position);

  **file pointer** - It is the pointer which points to the file.

  **displacement** -It is positive or negative.

- This is the number of bytes which are skipped backward (if negative) or forward (if positive) from the current position. This is attached with L because this is a long integer.

**Pointer position**: This sets the pointer position in the file.

| Value Pointer position | Value Pointer position |
|---|---|
| 0 | Beginning of file. |
| 1 | Current position |
| 2 | End of file |

**Example:**

**1. fseek( p,10L,0)**

- This 0 means pointer position is on beginning of the file, from this statement pointer position is skipped 10 bytes from the beginning of the file.

**2. fseek( p,5L,1)**

- This 1 means current position of the pointer position. From this statement pointer position is skipped 5 bytes forward from the current position.

**3. fseek(p,-5L,1):**

- From this statement pointer position is skipped 5 bytes backward from the current position.

| **Different When in fseek** | |
|---|---|
| **When** | **Meaning** |
| SEKK_SET | Starts the offset from the beginning of the file. |
| SEKK_END | Starts the offset from the end of the file. |
| SEKK_CUR | Starts the offset from the current location of the cursor in the file. |

**Program 2.26:** for fseek()

```
#include <stdio.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", num.n1, num.n2, num.n3);
}
fclose(fptr);
return 0;
}
```

**ftell()**

- This function is used to move the file pointer to the beginning of the given file. This function returns the value of the current pointer position in the file. The value is count from the beginning of the file.

  Syntax: ftell(fptr); fptr is a file pointer.

  **rewind()**

  Syntax: rewind( fptr); fptr is a file pointer.

**Program 2.27: program for fseek():**

**Write a program to read last 'n' characters of the file using appropriate file functions(Here we need fseek() and fgetc()).**

```
#include<stdio.h>
#include<conio.h>
void main()
{
FILE *fp;
char ch;
clrscr();
fp=fopen("file1.c", "r");
if(fp==NULL)
printf("file cannot be opened");
else
{
printf("Enter value of n to read last 'n' characters");
scanf("%d",&n);
fseek(fp,-n,2);
while((ch=fgetc(fp))!=EOF)
{
printf("%c\t",ch);
}
}
fclose(fp);
getch();
}
```

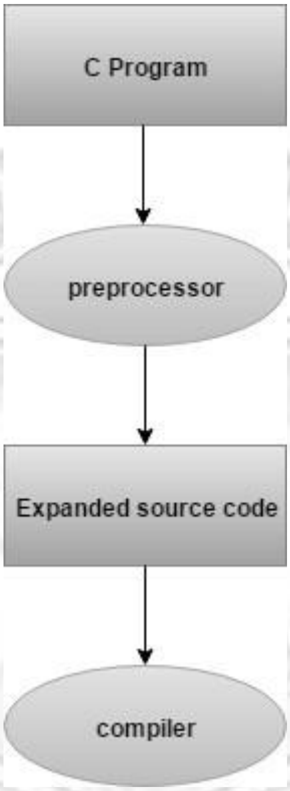
## PREPROCESSOR DIRECTIVES

- The C preprocessor is a microprocessor that is used by compiler to transform your code before compilation. It is called micro preprocessor because it allows us to add macros.
- ***Note:*** *A macro is a segment of code which is replaced by the value of macro. Macro is defined by #define directive.*

  **Example**

  ```
  #define PI 3.14
  ```

  Here, PI is the macro name which will be replaced by the value 3.14.



- All preprocessor directives starts with hash # symbol. Let's see a list of preprocessor directives.
  - #define: It substitutes a preprocessor using macro.
  - #include: It helps to insert a certain header from another file.
  - #undef: It undefines a certain preprocessor macro.
  - #ifdef: It returns true if a certain macro is defined.
  - #ifndef: It returns true if a certain macro is not defined.
  - #if, #elif, #else, and #endif: It tests the program using a certain condition; these directives can be nested too.
  - #line: It handles the line numbers on the errors and warnings. It can be used to change the line number and source files while generating output during compile time.
  - #error and #warning: It can be used for generating errors and warnings.

- #error can be performed to stop compilation.
- #warning is performed to continue compilation with messages in the console window.
- #region and #endregion: To define the sections of the code to make them more understandable and readable, we can use the region using expansion and collapse features.