File Handling in C

File:

- A file a collection of data stored in one unit, identified by filename.
- The primary purpose of a file is to keep record of data and record is the group of related fields.
- Each file ends with an end of file(EOF) at a specified byte number, recorded in FILE structure.
- A file must first be opened properly before it can be accessed for reading or writing. When a file is opened an object(buffer) is created and a file is associated with the buffer.
- In general, file is a collection of related records such as student's name, date of birth, address, marks, phone number etc.
- A file is nothing but a source of storing information permanently in the form of a sequence of bytes on a disk.

Buffer:

When the computer reads, the data move from the external device to memory and when it writes, the data move from memory to the external device. This data movement often uses a special work area known as buffer.

- A buffer is a temporary storage area that holds data while they are being transferred to or from memory.
- The primary purpose of a buffer is to synchronize the physical devices with a program's need.

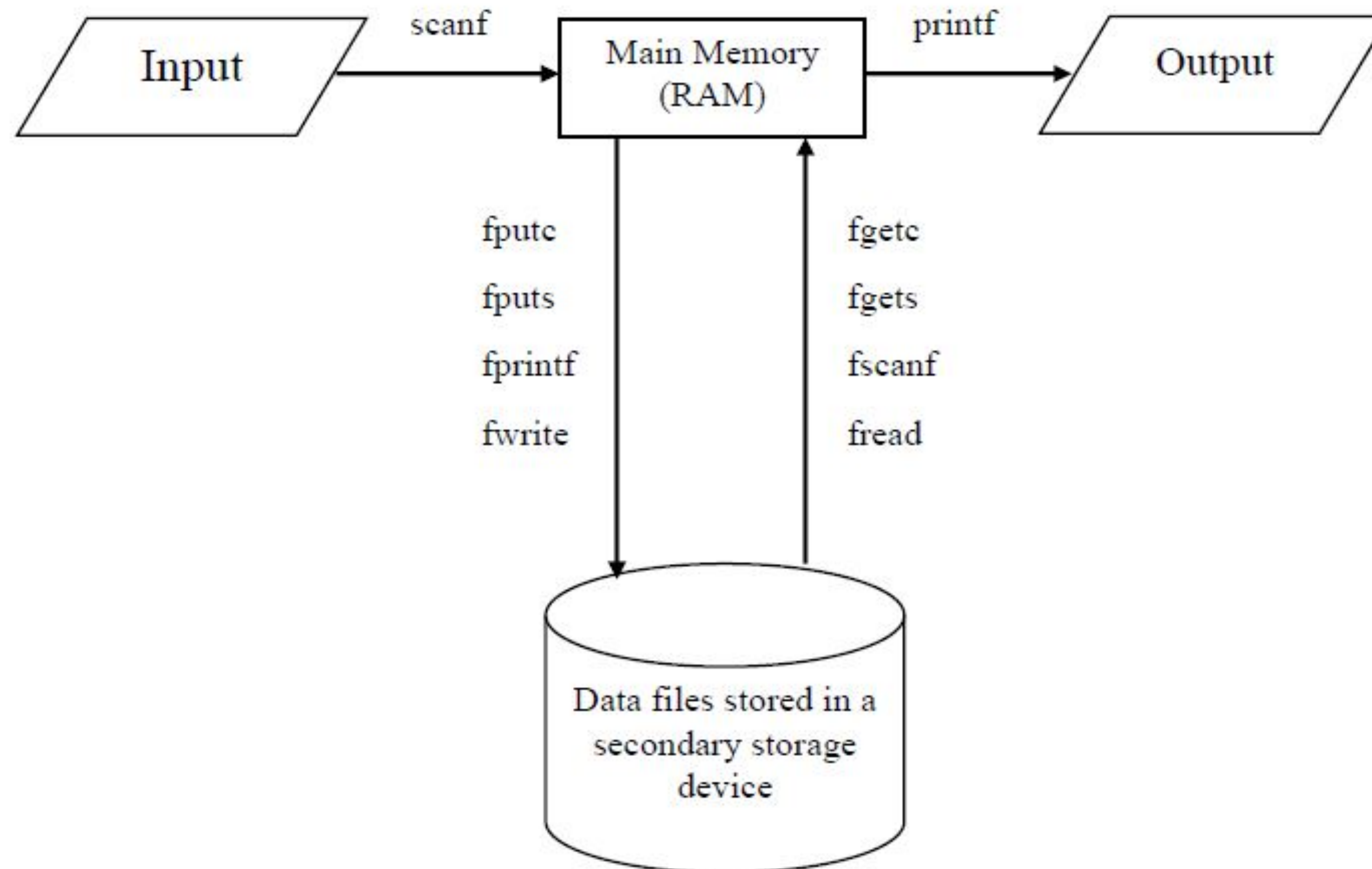
File Handling:

- File handling is the process of storing data in the form of input or output produced by running C programs in data file for future reference and analysis. And, we can extract/fetch data from a file to work with it in the program.
- File handling provides a mechanism to store the output of a program in a file and to perform various operations on it.
- File handling concept provides various operations like creating a file, opening a file, reading a file or manipulating data inside a file etc.

Why file handling?

- The data stored in the various of a program will be lost once the program is terminated because they are stored in the Random Access Memory(RAM) which is volatile memory.
- So, if we want store that data(input/output) used in the program permanently inside the secondary storage device so that, we can access these data from there whenever it is needed file handling concept is important.

Block diagram of file accessing using a various function written to a file and read from a file using various functions:



Types of file:

There are two types of file namely:

1. Text file
2. Binary file

3. Text file:

- These are the simplest files a user can create when dealing with file handling in C.
- It is created with a .txt extension using any simple text editor.
- A text file stores information in the form of ASCII characters internally, but when you open the text file, you would find the content of the text readable to humans.
- Text file consume large storage space.
- Example: new.txt, file.c etc.

2. Binary file:

- A binary file stores information in the form of the binary number system (0's and 1's) and hence occupies less storage space.
- In simple words, it stores information in the same way as the information is held in computer memory. Therefore, it proves to be much easier to access.
- Example: file.dat, photo.jpg, one.png, song.mp3, file.exe etc.

Difference between Text File and Binary File :-

Text File	Binary File
1. Output Formatted Text.	1. Output in 0 and 1 binary format.
2. EOF(End of File), \n(New Line) are used.	2. Binary mode not use such character.
3. fgetc ( ), fputc ( ), fprintf ( ), fscanf ( ), fgets ( ), fputs ( ) are used.	3. fread ( ) and fwrite ( ) functions are commonly used.
4. Numbers are stored as string.	4. Numbers are stored as int, float etc.
5. Takes more access time.	5. Takes less access time.
6. It occupies more memory than binary file.	6. It occupies less memory than text file.

Operations on File:

- Creating of a new File
- Opening an existing File
- Reading from a File
- Write to a File
- Moving to a specific location in a file
- Closing a File

File Operation Modes:

The different file mode that can be used in opening file are:

Mode	Description
“w” (write)	If the file doesn't exist then this mode creates a new file for writing, and if the file already exists then the previous data is erased and the new data entered is written to the file.
“r” (read)	This mode is used for opening an existing file for reading purpose only. The file to be opened must exist and the previous data of the file is not erased.
“a” (append)	If the file doesn't exist then this mode creates a new file and if the file already exists then the new data entered is appended at the end of existing data. In this mode, the data existing in the file is not erased as in “w” mode.
“w+” (write + read)	This mode is same as “w” mode but in this mode we can also read and modify the data. If the file doesn't exist then a new file is created and if the file exists then previous data is erased.
“r+” (read + write)	This mode is same as “r” mode but in this mode we can also write and modify existing data. The file to be opened must exist and the previous data of file is not erased. Since we can add new data and modify existing data so this mode is also called update mode.
“a+” (append + read)	This mode is same as the “a” mode but in this mode we can also read the data stored in the file. If the file doesn't exist, a new file is created and if the file already exists then new data is appended at the end of existing data. We cannot modify existing data in this mode.

File Operation Modes:

Continue…

Mode	Description
“wb”	Binary file opened in write mode.
“ab”	Binary file opened in append mode.
“rb”	Binary file opened in read mode.
“wb+”	Create a binary file for read/write.
“rb+”	Open a binary file for read/write.
“ab+”	Append a binary file for read/write.

Text mode

- In text mode every digit or text are stored as a character and while reading the content back, the conversion is required from character to appropriate format and takes lots of space.
- Character I/O, string I/O, and formatted I/O use text mode.
- If 3.14159 is stored in character mode file size would be 8 bytes (counts each character including decimal and EOF).

Binary mode

- In binary mode every digit or text is stored in binary format and while reading the content no conversion is necessary and takes little space.
- fread ( ) and fwrite ( ) are used in binary mode.
- If 3.14159 is stored in character mode file size would be 4 bytes.

Opening a file:

- A file must be opened before any I/O operations performed on that file. The process of establishing a connection between the program and file is called opening the file.
- A structure named FILE is defined in the file <stdio.h> that contains all information about the file like name, status, buffer size, current position, end of file status etc.
- All these details are hidden from the programming and the operating system takes care of all these things.
- A file pointer is a pointer to a structure of type FILE.
- Whenever a file is opened, a structure of type FILE is associated with it and a file pointer that points to this structure identifies this file.
- **Example**: FILE *fp;
- The function **fopen ( )** is used to open a file.
- **Syntax**: fp=fopen(“path:\\filename.ext”, “file_operation_mode”);

<u>Closing a file:</u>

- The file that was opened using fopen ( ) function must be closed when no more operations are to be performed on it.
- After closing the file, connection between file and program is broken.
- On closing the file, all the buffers associated with it are flushed i.e all the data that is in the buffer is written to the file.
- The buffers allocated by the system for the files are freed after the file is closed, so that these buffers can be available for other files.
- Although all the files are closed automatically when the program terminates, but sometimes it may be necessary to close the file by using fclose( ) function.
- **<u>Syntax</u>**: fclose(name_of_file_pointer);
- **<u>Example</u>**: fclose(fp);

End of file (EOF):

- The file reading function need to know the end of file so that they can stop reading.
- When the end of file is reached, the operating system sends an end-of-file signal to the program.
- When the program receives this signal, the file reading function returns EOF, which is a constant defined in the file <stdio.h> and its value is -1.
- This is only applicable to text mode file only.

File Processing Techniques:

The approach to read or to write the content from/to the file is know as file processing. There are basically two types of file processing techniques namely:

1. Sequential file processing:

In this type of files data is kept in sequential order if we want to read the last record of the file, we need to read all records before that record so it takes more time.

2. Random file processing:

In this type of files data can be read and modified randomly .If we want to read the last record we can read it directly. It takes less time when compared to sequential file.

Input/output Functions:

The functions used for file input/output are

1. Character I/O

fputc( ) :

- This function writes a character to the specified file at the current file position and then increments the file position pointer.
- Syntax: fputc(character,file_pointer);

fgetc( ) :

- This function reads a single character from a given file and increments the file pointer.
- Syntax: fgetc(file_pointer);

2. Integer I/O:

putw( ):

- This function writes an integer value to the file pointed to by file pointer.
- Syntax: putw(Integer,file_pointer);

getw( ):

- This function returns the integer value from the file associated with file pointer.
- Syntax: getw(file_pointer);

3. String I/O:

fputs():

- This function writes the null terminated string pointed by given character pointer to a file.
- Syntax: fputs(string,file_pointer);

fgets():

- This function is used to read characters from a file and these characters are stored in the string pointed by a character pointer.
- Syntax: fgets(string,length_of_string,file_pointer);

Formatted Input/output Functions:

fprintf( )

-This function is same as the printf( ) function but it writes formatted data into the file instead of the standard output(screen).

-This function has same parameters as in printf( ) but it has one additional parameter which is a pointer of FILE type, that points to the file to which the output is to be written.

Syntax: fprintf(fp, “control_string”,list of variables);

Example: Program used to write name, roll no and marks of student using fprintf function.

```
int main()
{
    FILE *fp;
    char name[20];
    int rollno;
    float marks;
    fp=fopen("std.txt","w");
    if(fp==NULL)
    {
    printf("File can not open.");
    }
    printf("Enter Name: ");
    gets(name);
    printf("Enter roll.no: ");
    scanf("%d",&rollno);
    printf("Enter marks: ");
    scanf("%f",&marks);
    printf("Now writing data into file.....");
    fprintf(fp,“Name=%s\nRoll.no=%d\nMarks=%.1f",
    name,rollno,marks);
    fclose(fp);
    getch();
    return 0;
}
```

fscanf( )

- This function is similar to the scanf( ) function but it reads data from file instead of standard input, so it has one more parameter which is a pointer of FILE type and it points to the file from which data will be read.

Syntax: fscanf(fp, “control_string”, &list_of_variables);

Example: Program to read data from a file using fscanf function.

```
int main()
{
    FILE *fp;
    char name[20];
    int rollno;
    float marks;
    fp=fopen("std.txt","r");
    if(fp==NULL)
    {
        printf("File can not open.");
    }
    fscanf(fp,"%s %d %f",name,&rollno,&marks);
    printf("%s %d %.1f",name,rollno,marks);
    fclose(fp);
}
```

Random access in a file:

- Random access means you can move to any part of a file and read or write data from it without having to read through the entire file.
- In this technique the content are stored sequentially and read back the content from any position of the file. And it can be performed using some function like

1. fseek( )
2. rewind( )
3. ftell( )

1. fseek ( ):

- This function is used to move file position to a desired location within a file.
- Syntax: **int fseek(fp, long int offset, int position);**

  - where, fp is a file pointer of type FILE which identifies the file.

  - The offset specifies the number of positions(bytes) to be moved from the location specified by position.

  - The offset may be positive means move forward or negative means move backwards.

  - The value of the positon may be 0 or 1 or 2.

Value	Constant	Meaning
0	SEEK_SET	Beginning of the file
1	SEEK_CUR	Current position
2	SEEK_END	End of the file

Continue…

Function	Meaning
fseek(fp,n,SEEK_CUR)	Set file position ahead from the current position by n bytes.
fseek(fp,-n,SEEK_CUR)	Set file position back from the current position by n bytes.
fseek(fp,0,SEEK_END)	Set file position to the end of the file.
fseek(fp,0,SEEK_SET)	Set file position to the beginning of the file.

//Example of fseek( ):

```
#include<stdio.h>
#include<conio.h>
void main()
{
	FILE *fp;
	char str[30];
	fp=fopen("abc.txt","r");
	if(fp==NULL)
	{
		printf("File can not open.");
	}
	else
	{
		fgets(str,29,fp);
		printf("%s\n",str);
		fseek(fp,3,SEEK_SET);
		fgets(str,29,fp);
		printf("%s\n",str);
		fseek(fp,-3,SEEK_CUR);
		fgets(str,29,fp);
		printf("%s\n",str);
		fseek(fp,-4,SEEK_END);
		fgets(str,29,fp);
		printf("%s",str);
	}
	getch();
}
```

2. ftell ( ):

- This function is used to get the current position of the file pointer.
- **Syntax**: **int ftell(FILE *fp);**

  - This function will return the relative offset(bytes) of the current position and fp is the file pointer.

  - This function will tell us that how many bytes already been read or written.

//Example of ftell( ):

```
#include<stdio.h>
#include<conio.h>
void main()
{
	FILE *fp;
	char str[30];
	fp=fopen("abc.txt","r");
	if(fp==NULL)
	{
		printf("File can not open.");
	}
	else
	{
		fseek(fp,-3,SEEK_END);
		int n=ftell(fp);
		printf("Current position of the file pointer is %d",n);
	}
	getch();
}
```

3. rewind ( ):

- This function is used to set the file pointer at the beginning of the file.
- **Syntax: void rewind(FILE *fp);**

    - Where, fp is the file pointer which identify the file.

Example of rewind( ):

```
#include<stdio.h>
#include<conio.h>
void main()
{
	FILE *fp;
	char str[30];
	fp=fopen("abc.txt","r");
	if(fp==NULL)
	{
		printf("File can not open.");
	}
	else
	{
		fseek(fp,-3,SEEK_END);
		rewind(fp);
		int n=ftell(fp);
		printf("Current position of the file pointer is %d",n);
	}
	getch();
}
```

Deleting a File:

- We use remove () to delete a file.

Syntax: remove(“file_name”);

Example:

```
int main()
{
    remove("std.txt");
    printf("Your file is successfully deleted.");
    getch();
    return 0;
}
```

Renaming a File:

- We use rename ( ) to rename a file.

Syntax: rename(“existing_file_name”, “new_file_name”);

Example:

```
int main()
{
    rename("std.txt","std1.txt");
    printf("Your file is renamed successfully.");
    getch();
    return 0;
}
```

Error Handling in Files:

- It is possible that an error may occur during I/O operations on a file.
- It is quite common that errors may occur while reading data from a file in C or writing data to a file. For example, an error may arise due to the following:

  - A file is opened with an Invalid name.

  - When the data accessed is beyond the EOF character from the file.

  - When a file is opened for one purpose but trying to perform different operations.

  - When a hidden file is trying to open then we can also get an error.

  - If the file is write protected then also we can get an error.

  - To handle these type of error, C provides error handling functions and these are:

1. feof ( )

2. ferror ( )

1. feof ( ):

- This function is used to detect the end of file (eof) character in the file.
- This function returns non-zero if file pointer is on EOF otherwise it returns 0.
- Syntax: feof(file pointer variable);
- Example: feof(fp);

2. ferror ( ):

- This function is used to detect the errors while reading from the file or writing into the file.
- It returns a non-zero value when an error is occurred while performing read and write operations otherwise it returns zero.
- Syntax: ferror(file pointer variable);
- Example: ferror(fp);

fgetc() or getc(), fgets() and fscanf(): These functions are input functions which read formatted or unformatted data from the file.

fputc() or putc(), fputs() and fprintf(): These functions are output functions which write formatted or unformatted data to the file.

These functions are not suitable for reading or writing large amount of data (block of data) to/from a file. So we use two additional functions:

1. fwrite()
2. fread()

1. fwrite ( ):

- Used to write block or record (fixed length group of data) of data to a file. A record may be an array or a structure.

Syntax: fwrite(ptr, int size, int n, FILE *fp);

Where,

ptr : ptr is the reference (address) of an array or a structure stored in memory.

size : size is the total number of bytes to be written.

n : n is number of times a record will be written.

fp : fp is a file pointer where the records will be written in binary mode.

Example of fwrite ():

```
#include<stdio.h>
#include<conio.h>
struct Student
    {
        int roll;
        char name[25];
        float marks;
    };

    int main()
    {
        FILE *fp;
        char ch;
        struct Student S;

        fp = fopen("Student.txt","w");

        if(fp == NULL)
        {
            printf("\nFile Can't open.");

        }

do
        {
            printf("Enter Roll : ");
            scanf("%d",&S.roll);

            printf("Enter Name : ");
            scanf("%s",S.name);

            printf("Enter Marks : ");
            scanf("%f",&S.marks);

            fwrite(&S,sizeof(S),1,fp);

            printf("\nDo you want to add
    another data (y/n):\n ");
            ch = getche();

        }while(ch=='y' || ch=='Y');

        printf("\nData written successfully...");

        fclose(fp);
        getch();
        return 0;
    }
```

2. fread ( ):

-Used to read block or record (fixed length group of data) of data from a file.

Syntax: fread(ptr, int size, int n, FILE *fp);

Where,

ptr : ptr is the reference of an array or a structure where data will be stored after reading.

size : size is the total number of bytes to be read from file.

n : is number of times a record will be read.

fp: fp is a file pointer from where the records will be read.

Example of fread():

```
#include<stdio.h>
#include<conio.h>
struct Student
      {

            char name[25];
            int roll;
            float marks;
      };
     int main()
      {
            FILE *fp;
            struct Student S;
fp = fopen("Student.txt","r");
     if(fp == NULL)
            {
                  printf("File Can't open.");

            }
            fread(&S,sizeof(S),1,fp);
            printf("Name=%s\n",S.name);
            printf("Roll NO=%d\n",S.roll);
            printf("Marks=%.1f\n",S.marks);
       fclose(fp);
       getch();
            return 0;
      }
```

Simple Example of append mode:

```
#include<stdio.h>
#include<conio.h>
#include<string.h>
int main()
{
	FILE *fp;
	int i;
	char ch[50]={"Kathmandu BernHardt"};
	fp=fopen("new.txt","a");
	if(fp==NULL)
	{
		printf("File can not open.");
	}
	for(i=0;i<strlen(ch);i++)
	{
		fputc(ch[i],fp);
	}
	fclose(fp);
	getch();
	return 0;
}
```

Q. Given a text file, create another text file deleting the following words "three", "bad" and "time".

```
#include<stdio.h>
#include<conio.h>
#include<string.h>
int main()
{
FILE *fp, *fpp;
Char C[10];
fp=fopen("file.txt", "r");
if(fp==NULL)
{
printf("File Can not open.");
}
fpp=fopen("text.txt", "w");
if(fpp==NULL)
{
    printf("File can not open.");
}
While(fscanf(fp, "%s", &c)!=EOF)
{
if(((strcmp(c,"three")!=0 && (strcmp(c,"bad")!=0 && (strcmp(c, "time")!=0))
{
    fprintf(fpp, "%s",c);
}
fclose(fp);
fclose(fpp);
getch();
return 0;
}
```