Syllabus:

UNIT-V

Input and output

Concept of a file, streams, textfiles and binary files, file input/output functions (standard library input/output functions for files), error handling, positioning functions (fseek, rewind and ftell).

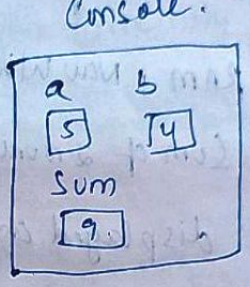
UNIT-V

File handling in C: (Input and Output)

↳ Introduction to file handling in C.
Why we need files in C.
What is exactly mean by a file

Concept of file: What is the need for file?

Whenever you write a program, you run the program, suppose you are adding 2 numbers.

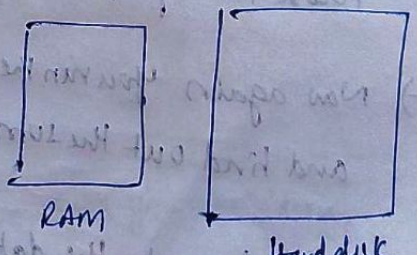


We have taken 2 numbers 'a' and 'b' and another variable sum

→ We are adding two no's and storing the value in sum variable and we could see this on console.

→ Console is the output screen, where the output will be displayed.

→ and whenever we run the program, the program would be loaded into the RAM memory. and another memory is your hard disk



→ RAM is a volatile memory, means. Whatever will be there in RAM, whenever you switch off your computer that will be gone.

→ Harddisk is a Non-volatile memory / you can say permanent memory. you will store anything in this memory and if you switch off your computer and whenever you turn it on, automatically that data would be there.

→ may be you have created a word document, and you saved it, you might have stored images, many data you might have stored in your computer.

→ But whenever you switch off computer and turn it on again, this data remains.

This is what we can say the data is stored into hard disk. (permanent storage).

→ But when you run a program that will be loaded into the RAM, Now whatever the values you enter, like if you take sum of 2 numbers, 5 & 4 and sum = 9, these will be displayed on console.

→ But after terminating of this program the data on the console will be lost, because it is in RAM.

→ Whatever we are executing, that data is now in RAM memory. and it is volatile memory, whenever you terminate / close that program, those variables, that values will be lost.

→ Now again you run the program, again you take new values and find out the sum. (This is simple example.

⇒ suppose i want the data should be stored somewhere, like whatever i have taken previously 5, 4 and sum = 9. these i want to store, such that after terminating / closing the program still this result / this data is there anywhere (need not know where it should be, but in memory it should be there). and if i need these values in future

whenever we run this program or any other program and then i need this data, so that i can fetch this data.

→ so i want to store this data permanently and where we can store?

obviously in hard disk. and harddisk is a permanent storage.

and how we will store this data in Harddisk?

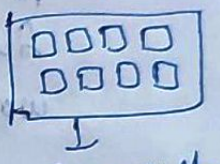
With the help of files.

What are files?

"Files are nothing but, sequence of bytes" (memory / blocks) Where you can store the data permenantly.

(streams).

In file → we will have sequence of bytes only.



memory is there → it's like a memory only.

and in these bytes we can store the data permenantly.

Files are used to store the data in Hard disk.

Why we need files?

After termination of the program, still you want to store the data, then we use files.

→ Next thing is suppose you want to enter large amount of data, it is not easy to enter that amount of data, so if that data is stored in a file permanently in hard disk, so you can access this file with the help of few commands.

→ you can write down few lines in your 'c' program and you will easily access these file.

→ Rather than entering that data, because it would be a tedious task, when you run a program, suppose a program it's something like that, you can easily fetch the data from a file and that's it.

→ And this will be considered as input of the program.

Let's say if the input of the program is large or very large, so rather than typing/entering the data by yourself, suppose that data is in a file, it would be easy to fetch that file, you have to just write down few commands and that would be considend as input.

→ And obviously for storing permanently the o/p we use files.

⇒ Obviously the RAM size is smaller than Harddisk. So Ram cannot handle large amount of data, so for that thing we need files.

So above all are the reasons why we need files in C.

What is file handling?

You have to handle these files, suppose in a program you need a data that is in a file, so you need to fetch that from RAM, so fetching or writing into the file or reading that file or opening that file, creating a file, this is known as file handling.

We are creating a file and if there is an existing file, we are opening that file, you are reading something from a file, you are writing something into a file, you are closing that file, that thing is called file handling.

Types of files:

Files are of two types:

1. Text files
2. Binary files.

Text files will have ".txt" extension.
Binary files will have ".bin" extension.

Text files:

These are simple plain text files like, you can write down anything like alphabets, symbols, numbers, those are what? with the simple text editor you can create these files like "notepad" or wordpad or microsoft word;

→ you can easily read these files, you can easily manage these files, but these are less secured.

Binary files:

→ Here, the data is stored in the stream of 0's and 1's.
→ So these are more secured.
→ obviously you cannot read easily this data.
→ so this is more secured than text files.
→ Generally these are the compiled files of these text files.

We can see in our computer also, .bin files also will be present

→ you can open it and you can just see what is there in those files → If you are able to see those files or not.

Note:

If data is in the form of files you can easily transfer that data from one machine to another machine.

Operations you can perform on files:

1. create a file using C program
2. open a file with C program's.
3. Read a file
4. write into the file.
5. close the file. ☺

These are the main operations on files.

We have many more operations on files too.

We shall understand all the operations of files, with the proper programs.

How we can open a file, how we can read that data, how we can close the file, How we can write something to the file, how you can append something to the file with the help of programs we will understand.

To handle these files we need a pointer and that is a file pointer. (You can say i.e. one use of pointers).

→ When you are handling files, you definitely need a pointer.

→ The pointer type should be "FILE" (capital letters only and this we can say as "datatype".

→ what is the prototype of "FILE" datatype, and this is already defined in that "stdio.h" header file.

→ so one pointer we need to handle these files

```
FILE *ptr;
```

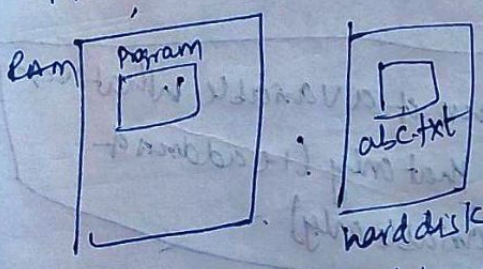
⇒ This is a file pointer.

datatype & pointervariable.

⇒ We need to understand more terminologies / terms / functions / pointers that will be used in file handling concepts.

→ Whenever you write a program / file handling program in C, you cannot write down this program without that functions or without the pointer or without the terms that we need to understand.

⇒ Whenever you execute your program then that will be loaded into the RAM (some memory will be given to that program in main() and we have a harddisk which is non volatile memory and RAM is a volatile memory.



If you want that whatever you are processing the input data and the output data to be stored permanently then using files we can store them in "harddisk".

And file we can say is a memory block or you can say a sequence of bytes where you can store more data permanently on hard disk.

⇒ suppose in harddisk you have a file named "abc.txt" and if you want to perform some operations on this file

like you can create a file, you can open a file, you can read this file whatever is there in this file, you can write something into this file and append something into this file (many operations we can do).

→ But if you want to perform operations on this file "abc.txt", you cannot directly access this file, this file needs to be first of all loaded into RAM (you can say a copy of this file would be loaded in memory while processing, so you cannot directly access this file just like that.

Imp

1. → you need a special pointer. ⇒ Which is a file pointer.

Suppose: we write `int *ptr;` ⇒ ptr is going to store address of a variable whose datatype is integer.

We know pointer is a variable that stores the address of another variable.

```
int a;
int *ptr;
ptr = &a;
```

⇒ General declaration of pointer.

The pointer can store the address of a variable whatever the datatype you specify that only (i.e address of that variable only).

⇒ I want to access this "abc.txt" file / process this "abc.txt" file or if i want to perform some operations with this file, so the pointer which is going to point to this file that would be what?

of this file type ? yes.

Because we cannot take like int pointer / float pointer these are primitive datatypes

so the pointer datatype here should be "FILE"

Here, we have a datatype "FILE" (Capitals) and it has been already defined in "stdio.h" headerfile. and this has been already created using how-? like below.

What is there in this implementation of data type.

typedef struct
{ ⇒ actually it depends on the operating system. here we will have a pointer (machine to machine) and the pointer in this is going to point to a file. buffersize and many variables.
} FILE; ⇒ this has been already defined in stdio.h

and you can directly use this datatype in your program. you just have to include this headerfile that's it.

→ suppose we are going to create a FILE type of pointer.

In the program we will create:

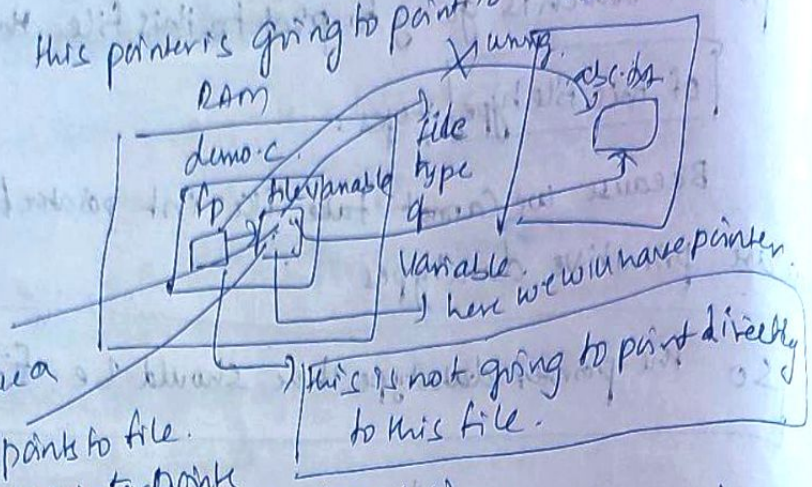
FILE *fp; → this pointer will point to / will contain address of file type of variable.

fp contains address of a variable whose datatype is FILE.

** so ultimately, actually we think that it is going to point directly to this file. (whatever the file). but behind the scene (abc.txt) process is what-).

Behind the scene, process is what?

FILE *fp; → this pointer is going to point to harddisk



here we have a pointer which points to file. (i.e. this pointer points to this file "abc.txt")

fp is going to point to a "file type of variable. (i.e. following typedef (pointer) variable).

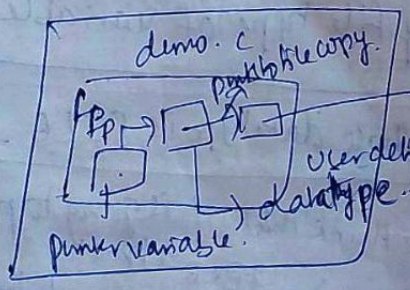
typedef struct
{ all have same interface pointer making like
buffersize.
:soon.
} FILE;

System dependent implementation

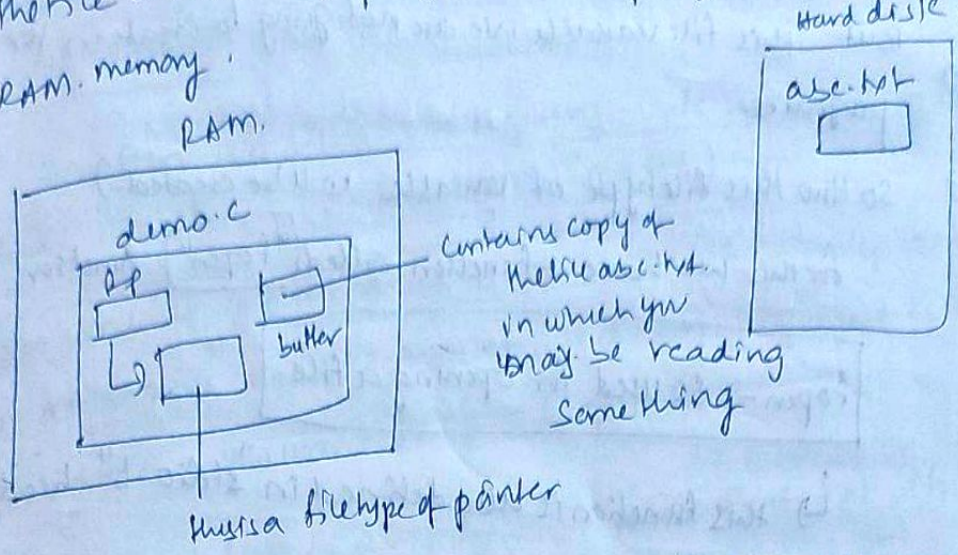
→ data type.
definitely we will have a pointer here which is going to point to the file abc.txt.

→ so more specifically if we say actually not to this file (abc.txt) because this is in harddisk.

This file first of all will be copied in buffer of program i.e. in RAM. and file is going to point to this buffer. (copy in RAM)



→ Because whenever we execute this program, obviously the copy of the file will be in program and program would be in RAM memory.



→ In the copy of a file, you may be reading something writing something or you may be appending something. and this will be reflected in original file abc.txt" here.

→ If you reading something in the copy of abc.txt file then obviously suppose in abc.txt I have typed name as Lakshmi and stored in hard disk and once you open that file the copy will be in RAM in that Lakshmi will be seen, this you can read this and print on your o/p screen.

→ If you want to write something then you will write it in copy of file i.e. When file is in buffer, let us say I write "Bhargav" and that will be loaded into hard disk finally after saving.

*** → But we need not move in such deeper process, you can just think like we have a pointer and this pointer is going to access the file. and using this file pointer only we can access this file, read/write/update.

fopen:

`File *fp;` ⇒ We are going to create in the program

But this file variable we are not going to create in the program.

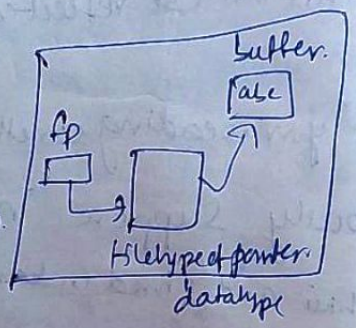
→ So how this file type of variable will be created?

for this we have a function called "fopen" function.

fopen → is used for opening a file.

↳ this function is also defined in "stdio.h" header file.

→ Whenever you call this function, it will automatically create this file type of pointer, and it will return address and this address is going to be stored into the fp, so fp is going to be pointed to the "filetypeofpointer" and this file pointer and buffer where copy of file is present (and these 2 tasks will be performed by fopen function.



It will create a file type of pointer and as well as a copy of file (abc.txt) will be opened in the buffer of RAM (here).

→ ultimately this fopen function returns address and that address will be stored in this "fp" and this "fp" will point to "filetype of variable" in which we have a pointer which points to this buffer (where copy of file is there) and ultimately using this we can access the actual file.

So this is the behind the scene process.

But we need not go into such deep process.

→ We can directly go into:

FILE *fp; → is going to point to the original file.

→ fopen is a function, so what are the arguments we are going to pass here. fopen("filename", "mode")

fopen → means you are going to open a file, so simply we will pass a "filename" as first argument and next argument is in which "mode" you want to open this file.

→ suppose in hard disk, I already have a file abc.txt, if you want to open this file, so simply pass:

fopen(abc.txt, "mode") Basic → 6 modes are there to open a file.

↓ you can even pass even the complete path.

Modes of fopen: (basic)

r → if you want your file as read only then we use "r"
w → If you want to write something "w"
a → If you want to append something use "a"
r+ → If you want to read and write both then use "r+"
w+ → If you want to read and write both but something different from r+ we use "w+".
~~a+~~
a+ → opens a file in both read and write mode.

Every modes we will discuss with separate programs.

xxvms
⇒ fopen function will return a pointer / we can say an address of file object / file variable / file type of variable and this will be stored in a pointer and obviously it will be a file pointer that would be "fp".

`fp = fopen("filename", "mode")`

↳ so with the help of this line the complete process we have explained will be done of accessing a file.

Now with the help of this "fp" (pointer) we can access the original file from harddisk. and we can read, write, append whatever you want to do and with the help of the mode whatever the mode you have written, according to that mode you can do something with that file.

Imp ⇒ After doing everything with the file i.e. whatever you want to do with this file, you are supposed to close the file.

`fclose(fp);` ⇒ It is the responsibility of the programmer to do this.

If you have opened a file, its responsibility of programmer to close the file.

⇒ and once you close the file, the buffer and the file variable (typedef ---) will be freed.

→ This means this file can be used by any other person. or by any other pointer.

→ We are not pointing to this file any more.

→ We have closed this file and it is very important and the buffer memory will be released (in RAM). and for this file variable memory also.

3 things are important:

1. Why we need a file pointer `FILE *fp;`
2. "fopen" function, why we need this & what is this
3. closing of a file using fclose function. `fclose(fp);`

The above three things are important for every program we write for file handling. (We have to use the above three things in every program of file handling).

Note!

If you want to read something from a file, we need to understand few operations like: fprintf, fscanf, fputc, fgetc, fgets.

fputs

fprintf => If you want to write something to a file we use this.
fscanf => If you want to read something from a file we use this.
fputc => If you want to write a character to the file we use this.
fgetc => If you want to read a character from a file we use this.
fgets => If you want to read a string from a file we use fgets.
fputs => If you want to write a string to a file we use fputs.

All the modes of the files and all the functions of file handling we shall understand in detail and with example programs one by one.

File output functions

How to write into a file in C:

⇒ How to write a character to a file.
⇒ How to write a string to a file
⇒ How to write a number to a file

We can do many things with a file: (functions)

→ We can create a file
→ We can open a file
→ We can read something from a file
→ We can write something to a file
→ We can append to a file
→ 6 modes are there (r, w, a, r+, w+, a+).
→ functions like fopen, fclose (functions which are used to read and write something to a file) fprintf — so on.

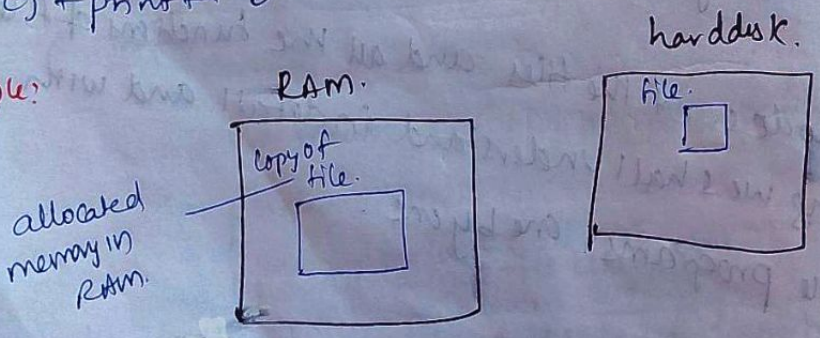
For writing into a file:

We have 3 functions for writing into a file.

a) fputc
b) fputs
c) fprintf.

} We will understand these functions and what are the differences

Example:



→ We have a hard disk and whenever we execute or run a program, in the main memory (RAM), some memory will be allocated to this program in RAM.

→ We want to write something into a file: So 2 cases can be there.

1. file exists
2. file doesn't exists.

File exists:

→ If the file is existing means it will be there in hard disk.

→ I want to write something into the existing file.

Then what we need to do →

a) We need to create a copy of this file / load this file into the RAM and this is what we can call it as "buffer"

b) We write into this buffer by writing some commands and then after that it will be updated to the original file present in hard disk.

File doesn't exist:

→ If the file doesn't exist ie. we don't have any file created in harddisk.

a) first of all we are going to create a file.

b) then we open this file and copy of this file is created in the RAM / loaded into the RAM

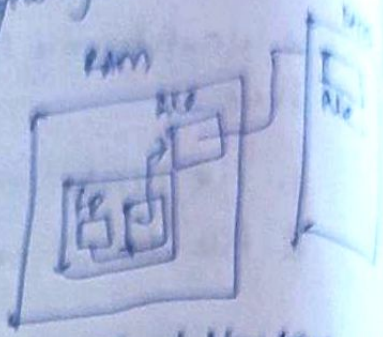
c) Then we will write into this file.

d) After this it will be updated to the original file on hard disk.

If we have to write into a file means, we are going to op... that file into write mode

→ first to access anything, to process anything with that file we need a file pointer

→ How we will create that pointer →)




```
main()
{
// In main() function we will write
    FILE *fp;   // This is file pointer and you can
                // initialize this pointer with NULL here.
    char ch='a';
// Now this fp will point to a filetype of object/variable and
// this filetype variable will have the address of the buffer
   (when copy of file is present) and then we can coordinate
   with the original file and this is the actual process.
// Now how to create this "filetype of variable" →?
```

and fp is going to contain the address of this "filetype of variable" and this process will be created when we use "fopen" function. and in simple terms we can say opening a file.

```
fp = fopen("demo.txt", "w");
```

write mode.

assuming that this file is in the same directory where I am writing the program. if so we don't need to specify the complete path, just specify the name of the file. if that file is at some other location then you have to specify the complete path. like (C:\---\demo.txt)

Now it will create "filetype of variable" and it will open this demo.txt file, i.e. it will load the copy of this file into the RAM. and it will return the address of this filetype of pointer variable in fp.

"w" mode:

What is this "w" mode?

a) "If file exists", then the previous content of this file will be erased and the copy will be opened in buffer of RAM.

And now whatever you write that will be returned to the original file now.

Like it is type of "overwritten"

The previous content is erased and fresh content is written.

b) "If file doesn't exists"

In this case, first a file is created, then it will open that file, means it will load that copy of file into RAM and now you can write to the file and will be saved to original file latter.

These are the two cases of "w" mode.

File output functions:

Now three functions are there to write something into a file.

a) fputc → write a character into a file.
b) fputs → write a complete string into a file.
c) fprintf → means you want to write something to a file.

Now if you want to write down a single character into the file, we use fputc.

We have took a character at beginning (char ch = 'a'); and i want to write this character simply into this "demo.txt" file.

another way.

fputc(ch, fp); // fputc('a', fp);

↓ ↳ you can only access this file with the help of this file pointer

Syntax: function name (character, file pointer);

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

Now when you execute this thing 'a' would be returned to this file

complete program: using "fputc" to write into a file // a single character

// taking i/p from user

```
#include<stdio.h>
main()
{
    FILE *fp;
    char ch = 'a';
    fp = fopen("demo.txt", "w");
    fputc('a', fp);
    fclose(fp);
}
```

o/p: demo.txt

a.

```
main()
{ FILE * fp;
  char ch;
fp = fopen("demo.txt", "w");
printf("Enter ch");
scanf("%c", &ch);
fputc(ch, fp);
fclose(fp);
}
```

o/p: a will be stored in this file.

Now fputc will write down only a single character.

Suppose we want to write a complete string to the file, then what you will do?

or you can take the Input from the user.

writing a string to a file using 'fputs' function.

```
#include<stdio.h>
main()
{
    FILE *fp;
    char str[50] = "C programming";
    fp = fopen("demo2.txt", "w");
// Now if you want to write a string to file we can have.
// 2 cases here.
```

simply you can use "fputs" and the syntax of "fputs" is:

```
fputs("c programming", fp); // directly writing string
fputs(str, fp);  // we are writing a string c programming
                 // and this string is stored into str (name of
                 // the variable.
fclose(fp);      // now it will write down the complete string
                 // into demo2.txt file.
}
```

olp : demo2.txt

c programming.

writing a string to file using fputc function.

If you want to use putc function, to write a string to a file. we have to store character by character and for this we use loop.

```
main()
{ FILE *fp;
  char str[50] = "C programming";
  fp = fopen("demo3.txt", "w");
  for(i=0; i != strlen(str); i++)
    fputc(str[i], fp);
    fclose(fp);
}
```

// string is a null terminated character array and by default the compiler will append \0 at the end of string.

// string is always terminated by '\0'

// strlen doesn't count Null character.

olp

demo3.txt

c programming.

Why we use fprintf function?

fprintf function - means it is formatted output, it is used for a string or a character, it is used for a mixed datatype like we can write down numbers also, string also, character also, float also. / just we have to specify the format specifier of corresponding number, string, character or float. ie why it is known as formatted output function

I want to store a number and a string to a file.

```
main()
{
    FILE *fp;
    int a=10;
    char str[50]="C programming";
fp= fopen("demo4.txt","w");
// using a single fprintf function we can write down a
   number also and a string also to a file.
   fprintf(fp,"%d %s",a,str); // writing a
            ↓                    // no and string to a file.
        file pointer.
   fclose(fp);
}
```

o/p: demo4

```
10
C programming
```

// Runtime initialization of values is assignment to students.

Note!

Suppose fopen is not able to open / create a file which is already existing due to something / some problem.

Then what will be returned by this "fopen" function?

fp now will contain NULL value

so before using the logic, check.

```
if(fp==NULL)
    printf("no able");
    exit(1); // Exit from there.
```

```
#include<stdlib.h> //for exit(1).
main()
{ FILE *fp=NULL;
  int a=10;
  char str[30]="C programming";
fp=fopen("demos.txt","w");
  if(fp==NULL)          // if something is there in fp means
  printf("no file");    // the file has successfully been opened
  exit(1);              // so that fp is having pointer to the's
                        // file type of variable
fprintf(fp,"%d %s",a,str);   o/p  demos
                                  10 C programming
  fclose(fp);
}
```

// Note: When we are executing this programs, files we dont have this demo.txt ---- so on all files will not be there in the directory

When we have written this C programs.

⇒ and if you open the files "demo.txt", --- so on any file, if you open them in write mode, then this "demo.txt" will be created first and then opened.

write this code in every program.

```
if(fp==NULL)
{ printf("error");
  exit(1);
}
```

File Input functions:

How to read from a file in C:

→ How to read the content of a file.

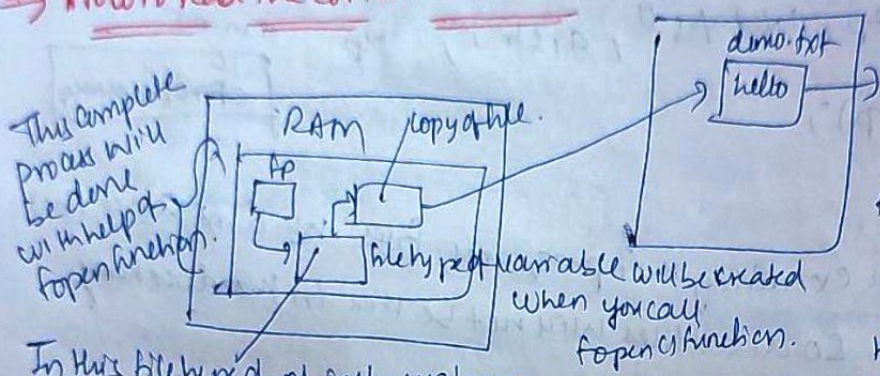
→ We will discuss read mode.

→ The functions which are used to read the contents are:

fgetc: for reading a single character.

fgets: for reading a complete string.

⇒ How to read the Content of a file:



In this filehypted object, we have some variables and then we have a pointer also and that pointer is going to contain address of buffer → Using this we can read file.

So this means one copy of this file would be opened in RAM (Buffer) using fopen function.

→ Whenever you read the content of "demo.txt" file, that content will not be erased from this file.

→ We are just reading the copy of this content.

File input functions:

fgetc() function and fgets() functions.

fgetc() function is there for reading i.e. we are getting from the file (reading).

→ fgetc function reads a single character

→ First of all the file typed variable contains a pointer and this pointer will contain the address of first byte / base address of the file.
This means it will read the first character means it will read 'h'.

⇒ If you are using fgets() function, then it will read the entire string.

How these functions will be returned and the process behind it is what →

```
#include<stdio.h>
main()
{ FILE *fp = NULL; // Creating a file pointer and assigning
                   // it with NULL.
```

// First we are will use a fgetc function, we will read a single character
⇒ from a file, so what character you read, that character you will obviously will store somewhere, so we will take a character typed variable.

```
char ch; // we took a character type of variable, char ch
         in this ch we are going to read the character.
```

// When you are going to execute the program, that program will be loaded into the main memory, some memory will be allocated to that program, the variables will be created and memory will be assigned to those variables and one variable will be ch and in this ch we are going to read the character.

→ using this ch, we are going to print the read character onto the output screen.

```
fp=fopen("demo.txt","r"); // we are opening the file in a read
// mode, in this case, if you want to read something from a file
```

prerequisites: That file must exist on the computer.

→ If there is no such file like "demo.txt", it will give e... and whatever the fopen will return, the address will be st... in fp

Vimp.
→ If suppose this file is not there in the computer then in "fp" it will have NULL.

So we have to check this condition in the program.

```
if(fp==NULL)
{
  printf("error");
  exit(1);
}
```

Note:- If fopen is not going to open this file because of some error or if the file doesn't exist, so fp contains NULL as fopen returns null and then if fp==NULL, it will print the error message and will exit.

⇒ if suppose, successfully this file has opened using this fopen function, then what will happen? Logic follows. first of all we will read character by character, so we have to write like below:

```
ch = fgetc(fp);   // use fgetc and just pass the file pointer and whatever this function is going to return it will be stored in ch.
printf("%c", ch);
fclose();
}
```

Op: Now it will simply print 'h' on the output screen

Complete program to read the contents of a file using fgetc() (using fgetc() function, we can read only a single character

```
#include<stdio.h>
int main()
{ FILE *fp = NULL;
  char ch;
  fp = fopen("demo.txt", "r");
  if(fp==NULL)
    { printf("Error reading file");
      exit(1)
    }
  ch = fgetc(fp);
  printf("%c", ch);
  fclose();
}
```

demo.txt

hello

O/p: h

If you want to read the entire string:

Suppose in my file i have: "C programming" and we want to read this complete string and if you use "fgetc" function i.e. character by character you want to read then you have to use loops and until this fgetc function and till when you want to read -> end of file.

Reading the Entire string using fgetc() function:

```
main()
{ FILE *fp = NULL;
  char ch;
  fp = fopen("demo.txt", "r");
```

```
if (fp==NULL)
{ printf("Error reading the file");
  exit(1);
}
while ( !feof(fp))
```

↳ just pass the pointer as we can access the file through this pointer only.

predefined function which indicates end of file

↳ but this is not the correct syntax:

this "feof()" ⇒ function will return false, if this is not the end of file.

so first we are at C ⇒ is it end of file → No, so it will return '0'.

but while(0) ⇒ means the condition is false so we will not enter into the loop, so how you will read the content?

so we have to modify this condition and we have to put logical not operator. (negation operator we can say), so we write as:

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

as while(0) is false so we used ! operator.

at last when it reaches end of the file feof() function returns the 1-C. ! so !(1)=0 now the loop exits.

o/p: C programming.

Let us say the complete string you want to read and let us take the string as "hello welcome to classes"

Reading the Entire String using fgets() function:

How you use fgets() function inc?

→ In order to make the logic for this we will not make the while loop and after if (fp==NULL) and after exit(1) and closing of this if block we will write as:

⇒ fgets(here we require 3 arguments?
1. string that you want to read (it will read from a file) where you will store that string first of all.

// In the previous program we have read one character and we have stored in ch.

// so here inc will read a string and where we will store that string → obviously we need to declare a string in the program. Rather than ch, we will declare char str[20];

So in fgets() function ⇒ the first argument that we pass would be a string (where you want to store, whatever you read i.e. in str we will store the read string).

2nd, Next is in a complete string present in a file "how many characters you want to read at one time", suppose i would like to read 5 characters then you have to specify '5' and if your specifying this '5' then it will read 4 characters i.e. "hell" only and one character will be for '\0' (null) because "string is null terminated character array"

if you want to read complete "hello" then you have to specify '6' (one for null). i.e. the no of characters i.e the length like pass as "second argument.

3rd, will be file pointer (fp). and then print using printf("%s", str);

fgets will return str (pointer to a string).

So whatever we have read from file that will be in str.

and in this case after that fclose → ? what would be printed.

From the file only 5 characters means "hello" this would be stored in 'str' first and then it will print str so it will print hello. and this before printing will be stored in "str" first of all and this str we print.

```
printf("%s", str);
```

hell will be printed.

⇒ In file we have;

⇒ the complete string is "hello welcome to classes"

⇒ Now i want to read the complete thing.

// Increase the length according to this buffer only i.e. str[10] only

```
fgets(str, 10, fp);
```

↓

maximum length we can take is only 9. and one for null.

It will read only 9 characters, but after that if you want to read then we have to put this into the loop and till end of file it will read.

When it is going to stop?

3 conditions are there:

Condition 1:

```
fgets(str, 5, fp);
```
(5 → length)

If we write 5, it will read 4 characters and it will stop.

Suppose if 'n' is the length, then n-1 it will stop. and it will print the n-1 characters. and then it will stop.

Condition 2:

char str[50]; → we will take.

```
fgets(str, 45, fp);
```
// obviously our string according to this it should read 45 char's.

// "hello welcome to c lectures" is of not 45 characters.

and in the file we have less than 45 characters, so once it will read to the end of file, it will stop reading. (only 25 we have).

Condition 3:

suppose a file is something like below

```
Welcome Students
Let's Start C Lectures
Course all the best
```

```
fgets(str, 45, fp);
```
If you give like this and the text is like above then ie. we have three lines in this file and we have given 45, but it is going to stop here. ie. at first line end character '\n'. it will stop here.

⇒ once this function fgets finds a newline character ie. Let's Start C Lectures will start in new line.

It is going to stop.

Suppose in a file we have content like this a new line
and text in 3 lines.

Then how can we read the complete data / complete file

simply put `fgets(str, 45, fp);` in loop.

program to read string:

```
main()
{
  FILE *fp = NULL;
  char str[20];
  fp = fopen("demo.txt", "r");
  if(fp == NULL)
    {
      printf("Error");
      exit(1);
    }
  fgets(str, 45, fp); // this will read string in one line only.
  printf("%s", str);
  fclose();
}
```

Program to read a string when the content is in 3 or n lines. --- n lines.

```
main()
{
  FILE *fp = NULL;
```

```
char str[50];
fp = fopen("demo.txt", "r");
if(fp == NULL)
{
    printf("error");
    exit(1);
}
while(!feof(fp))
{
    fgets(str, 15, fp);  // I have given length is here.
    printf("%s", str);
}
fclose();
}
```

demo.txt.

Welcome students
Let us start
C lectures files.
concept.

So first of all it will read 15 characters which includes 14 characters with one null & it will again that thing and again it will check the loop and it is not end of file, because end of the file is the last line, so again it reads 10 characters which includes 14 + 10 till last line and prints them.

Op:

Complete demo.txt file contents will be printed with newlines also.

fscanf() function:

This is used for formatted input.

Note: When you are opening a file in read mode, if the file doesn't exist, it will give an error.
But if you are opening a file in write mode, if the file exists, it will open file and write to the file else if the file doesn't exists, it will create the file and it will open it.

Append mode in file handling:

Append mode → means a file is there in the computer and we have some contents of that file and you want to add more content in that existing file, then how you will add (or) how you will append?

For this you have to open the file in the append mode.

→ If you want to open the file in read mode means, you can just read the content, you cannot write something to a file, you cannot add something.

→ If you open the file in write mode, then obviously we can write something to a file, but the previous content which is there in the file, that will be erased, means the content will be overwritten.

But here the situation is, you want to keep the previous content, as well as, you want to add some more content to a file, then we use "append mode" to open a file.

→ We shall understand append mode with the help of a program.

// First of all we need to open this file and which mode we need to open? how you are going → append mode.

demo.txt

hello Students.

```
main()
{ FILE *fp = NULL;
  char str[50]; // whatever you want to append that will be taken into this string.
fp = fopen("demo.txt", "a"); // Opening the file in the append mode.
```

```
if (fp == NULL)
{ printf ("error");
  exit(1);
}
```

if this is not the case, if fopen has successfully opened this file, into the main memory then just ask from the user the content you want to append.

```
printf("enter the content you want to append");
gets(str);
```

// now how you will put the content into the file. simply we have the functions, for writing something into the file: fputc, fputs and fprintf. and among these functions whatever function you want to use you can use.

// fputc → will add only one character → so you can use this function using loop.

fputs → you can use fputs for a string (or) you can use fprintf.

// we are using fprintf */

```
fprintf(fp, "%s", str); // fputs(str, fp);
```

→ it will append in next line.

↳ it will append in same line.

```
printf("Content Successfully appended");
fclose(fp);
}
```

demo.txt

hello dear students.

op:

Now whenever you open the file using fopen function, the cursor will be at the end of the text line. i.e. in the last of the existing file, and from here you will add the content or you will append the content.

We have seen the logic of fprintf and fputs, and if you want to use fputc → you can add the while loop and do it.

or any loop.

Note:

If you are opening the file in write mode / append mode

Similarity: (same thing is what?)

a) If file exists: they will open the file

b) If file don't exists: both will create a new file.

dissimilarity:

a) In "w" mode, the content will be overwritten.

Suppose hello is there then i write students

then "hello" will be erased and "students will be overwritten.

b) In "a" mode, No content overwriting, both previous and new content both will be there

r+ mode (special mode).

What is "r+" mode?

"r" → We know that "r" mode → means we can read the content of a file if it exists.

→ and if file doesn't exists, it will give error; it will return "null" only, it is not going to create a new file like in "w" mode.

"w" → In "w" mode, as this is write mode, if the file doesn't exist it will create a file and open this file.

"r" → But in "r" if file already exists, it will open the file for reading else gives error if file doesn't exist

but in both cases we can just read, we cannot write.

"r+ mode"

This will open the file for both reading and writing, but the main purpose of "r+" mode is reading.

** Here if the file exists → it will open the file both for reading and writing, but if the file doesn't exists → it is not going to create a new file, this we need to take care.

Lets take one Example:

We have one file named "sample.txt" on my hard disk, I want to read the content of this file, so first of all we will create a file pointer and starting we initialize it with NULL.

```
FILE *fp = NULL;
```

using "fopen" function, we will open the file, just provide the name of the file "Sample.txt", "r+");

```
fp = fopen("sample.txt", "r+");
```

// Now we have to check, because of something fopen is not going to / not able to open this file, then it will return NULL.

```
if(fp == NULL)
{
    printf("Error opening file");
    exit(1);
}
```

Suppose in file i have the following content.

```
Hi, this is
awesome.
```

/* // Now if you want to read the contents of a file and you can write also. but main purpose is to read and we can read the contents of file using fgets, fgetc functions. and if you want to write somethings to this file, where you can write-) in append mode we can append at the last and if you open the file in write mode means, the previous content gets erased

but if you open the file in "r+" mode then where you can write?

↳ This is very important about "r+" mode.

This point makes this mode special.

We can

- You can write any where. ⇒ that is from starting, from middle, or at the end.
- Just we have to set your file pointer.

↳ the logic you can write and you can write any where, if you open the file in r+ mode.

Basically if there is something in a file and you want to modify, you don't want to add a new content, you don't want to append, but you want to modify the already existed content from that file.

Suppose ten lines are there (10 lines) and two or 3 lines you want to modify.

↳ In that case open the file in r+ mode.

you can modify the file like this using this r+ mode only.

→ No other mode will help you to modify the file.

So for modifying the file just open in r+ mode, so this is very special, so move your file pointer according to that and you can modify.

So Sample.txt file above is my file, and i want to put something here (in this file).

fputc("z", fp); ⇒ where this z would be returned ? at the place of H (in the starting), the cursor would be here at starting only and now H will not be there and here you can add z and the remaining content would be the same.

fputs("hello", fp); // now the file z(; this is awesome gets replaced with z hello is as awesome and the remaining content will be same.

fclose();
}

Obs: This is how you have modified the content.

and if you want to modify the content from same position, then just you have to use the file pointer. and according to that we have to specify the logic. but by default if you modify the content it will be added from starting only. and if you want to read then read just using fgetc or fgets.

Complete program:

```
#include<stdio.h>
void main()
{
  FILE *fp = NULL;
  fp = fopen("sample.txt", "r+");
  if(fp == NULL) { printf("Error"); exit(); }.
fputc("z", fp);
fputs("hello", fp);
  fclose();
}
```

→ w+ mode:

This mode is also for reading and writing.

What is the difference between "r+" and w+?

Both r+ and w+ are for reading and writing but there is huge difference between these two.

These differences we shall understand with the help of program.

What is w+ mode)

It will open the file for both reading and writing, but the

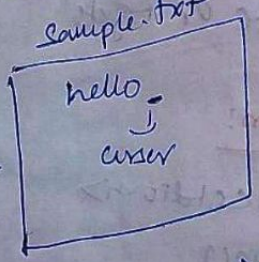
Special case is:

"If file doesn't exist", means in w+ mode, it will create new file. and we will open that file.

If file exists:

In this case, then it will "truncate" the file length to "0", means it will erase the previous content and the cursor will be at the starting of that line.

→ Now in sample.txt file, we have the content hello and the cursor will be at the last.



→ Now if file exists and you will open this file in w+ mode means this hello will be erased and the cursor will be at the starting only

→ Now the file length is "0".

→ This mode is specially for writing purpose.

– r+ mode was mainly for reading purpose, but we can do writing also.

Both "r+" and "w+" modes are used for reading and writing purpose, but in r+ mode if file doesn't exist, it will not create a new file and it will return null and that's it.

→ In w+ mode, if file doesn't exist, it will create a new file and opens that file.

→ In r+ mode, if file exists then it will open the file but previous content will not be erased, it would be as it is and your new content you can add whenever you want to add. may be in the starting or in middle of the already existing content or the last.

→ So mainly if you want to update the previous file then you will open that file in r+ mode.

→ But in the "w+" mode the previous content will be erased, so you can write down the new content from the starting only.

The above are the differences between w+ mode and r+ modes

Program for w+ mode:

```
#include <stdio.h>
void main()
{ FILE *fp = NULL;
  char str[20]; char ch;
  fp = fopen("sample.txt", "w+");
  if (fp == NULL)
  { printf("Can't open file");
    exit();
  }
```

sample.txt

hello..

If it doesn't exist it will create a file and open the file.

If due to some conditions fopen couldn't open the file then in that case it will print error and exit. It will return null

If fopen successfully opens the file, in that case, if file doesn't exist, it will create a new file and opens it.

→ Now, you can read the content, and the question here is, already in the file existing content is "hello" and wt mode is for both reading and writing.

→ So can you read this hello ? yes/no ?

VVVimp ↳ NO ⇒ Why so because you are opening this file in "w+" mode, so as soon as you open this in w+ mode as soon as this file would be loaded in that RAM, this hello will be erased. ; so you cannot read this content.

Then What content you will read - (in w+mode)

Whatever the content, you will write here now ! the new content), that content you can read..

This is not possible in write mode. i.e "w" mode.

you can only read the content, you cannot write — (true)

```
fputs( "world", fp);
```

↳ this i want to write, a string we are providing you can take the input from the user, many possibilities are there, we can try out. your own different different things.

So this "world" i want to put in the "sample.txt" file.

Now what will happen is hello will be replaced with world

Now at this point of time, the cursor will be at the last only.

Now you will say, I want to read this content, for reading we need to write a loop as below:

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

clearly how we are reading.

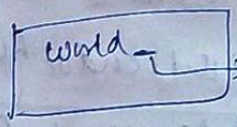
we have to write down a while loop and feof ⇒ means file end of file and "eof" will be returned by this function.

!feof(fp) ⇒ we are passing the file pointer to this function and a logical not.

Now is it possible to read this content in while loop → this means the program will print "world" or not → No it will not give.

Why so →

Because we have written fputs("world", fp); here in the program and now the cursor is at end of this in the file. i.e - file pointer is pointing to end of "World" in the file.



Now the condition is while(feof(fp)) ⇒ means it will return end of file. and yes 'fp' is at end of the file so it will return "true". means "1".

and while(!feof(fp)) ⇒ while(!(1)) ≡ while(0) means it is false, so we will not enter into this while loop and fclose so we are not able to read this content.

⇒ So what we should do now →

We have to move the "fp" from the last to beginning. to read the content.

P-T-O

⇒ How to move file :- rewind()

There is a special function called, "rewind"(fp);
after rewinding, just pass the file pointer.

so rewind function where we will write in the program :-

after writing fputs("word", fp); we will use this function rewind(fp);

Now :- whenever the compiler sees this rewind function, it means the file pointer is moved from last to first.

→ Now we can read this content. and now after rewind(fp); we can write the while logic of while(!feof(fp)); it will return "end of file" and the curser is at beginning only. and is it end of file, "no" so it will return false and false means '0' and !0 means true and it is 1

so :- while(1) is true. then it will enter the while loop and fgetc → means a single character we will read, means 'w' will be read from the file, and it will be stored in "ch" and 'w' should be printed.

→ Again the curser will be at first only and we go on read the text character by character and after it will return end of file and that's it.

→ so you are supposed to write down a special function in a program while using "w+" mode and while reading the content.

Complete program:

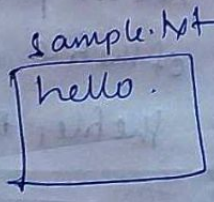
```
#include<stdio.h>
void main()
{
  FILE *fp=NULL;
  char str[20]; char ch;
  fp=fopen("sample.txt","w+");
  if(fp==NULL)
  { printf("cannot open file");
    exit(1);
  }
  fputs("world",fp);
  rewind(fp);
  while(!feof(fp))
  { ch=fgetc(fp);
    printf("%c",ch);
  }
  fclose();
}
```

"a+" mode:

→ We will understand what is this a+ mode
→ How it is different from r+ and w+ modes.

Now what is a+ mode?

Suppose i have a file "sample.txt" and in this file i have only "hello" text.



a+ → means we are opening this file both for reading and for appending. → reading / appending.

Not reading and writing it is reading and appending.

appending means => writing at the end of the file.
writing means => we can write at beginning or end or anywhere in the file. by moving your file pointer.

But basically we say "a+" mode will open the file in reading and appending mode.

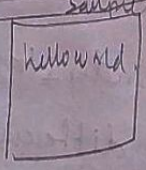
If you will read then, the cursor will be there at beginning for reading purpose and "hello" will be printed on the screen.

If you want to append something means the cursor will be there after hello and from here you can append from the end of the file. I at the end.

Appending means, basically we are writing at the end of file

So if i want to append "world" then that would be appended after hello in the file.

It doesn't mean that for reading it has one pointer that is at the beginning of the file and for appending it has another pointer at the end of the file, means it has basically 2 pointers -) No.



Only one pointer is there => It depends, if you are reading that pointer moves to beginning and if you are appending then cursor will be moved after hello [end of the file]

program:

```
#include<stdio.h>
void main()
{ FILE *fp=NULL; char ch;
fopen("sample.txt", "a"); // so it will open this file first
// for both reading and appending and if file doesn't exist
means (if no such file in my pc) then it will create a new
file and the new file will be opened.

if (fp==NULL)
  { printf("file doesn't exist");
    exit();
  }
// if you want to read first, you have to write the logic for reading first
and hello will be printed and if you want to append something
you can simply write down like below.

fputs ("world", fp); // world will be appended to hello
                        only.
// after the above if you want to read the logic is below.
    rewind (fp);
  while(!feof(fp))
    { ch=fgetc(fp);
      printf("%c",ch);
    }
  fclose();
}
```

What is the difference between r+, w+ and a+ ?

r+ ⇒ It is opened for reading and writing, but if file doesn't exist it will not create a new file. But in r+, the previous content will be as it is.

w+ ⇒ It will open a file for both reading and writing, but if file exists, one thing is the previous content would be erased, the new content whatever you write will be overwritten.

a+ ⇒ In a+ also if file doesn't exist, it will create a new file and it opens the file for reading and appending (writing the content will be appended at the last).

Note:

→ If you write fputs and rewind functions after while loop it will print hello only.

→ and after while loop if you write this fputs and rewind functions if you append "world", world will be appended after hello, but world will not be printed.

Error handling in files

→ Generally when we want to perform any I/O operations on files then we may get some errors and to handle those errors, we have to use some error handling functions.

Reasons why errs will occur?

1. When a file is opened with an invalid name then we may get err message.
2. When we are opening the file using read mode and when we try to perform write operations then it will give err message.

3. When we want to access a character which is beyond the end of the file character then we may get some error messages

Let the file contain 10 characters, so the last character of the file is "eof" but if you try to access 12th / 13th character then it will give error message.

4. We will be having some hidden files. Hidden files means we should not open those files from our program and if you try to open hidden files, then the compiler will give error messages.

These are some reasons why we get error messages

In order to handle the errors; mainly there are two functions are available:

1. feof() function
2. ferror() function.

feof() function: The name itself specifies the meaning.

It is used to check whether the character is the "end of the file character" or not.

→ If the character is "end of the file" character and it returns "true" value means '1'

→ Suppose if the character is not the "end of file" character then it returns false value.

Syntax:

```
feof(fp);
```

↳ file pointer.

What is the advantage of feof() function?

feof() function allows us to check whether the character is the "end of the file character" or not.

→ If the character is Eof character it returns true else false value.

2. ferror() function:-

It is useful inorder to check whether the file is successfully opened or not.

→ If there is an error in opening the file, then ferror function will return true value.

→ Suppose if the file is successfully opened, that means there is no error in opening the file, then it returns the false value.

Ex: fopen("sample.txt", "r");

↳ ferror will check whether this file successfully opened or not.

If this file doesn't exist, then ferror() function returns true value (means there is some error here).

→ If there is no error, means file is successfully opened. Then it returns false.

Programs illustrating the ferror() and feof() functions.

fopen("sample.txt","r");

```
#include<stdio.h>
main()
{ FILE *fp; char ch;
// In this example we have sample.txt file, and let us assume
// it contains some content and with the help of these two functions we have
// to read the data and display ie. print on the monitor.
fp = fopen("sample.txt","r");
// Now we have to check whether this file is successfully opened or not.
if (ferror(fp))                    // if we don't have any file then
printf("file not opened");         // ferror returns true and prints
                                   // the error message file not opened
else
{                                  // suppose fopen has executed and the file is successfully
                                   opened then we need to read the content.
while(!feof(fp))                   // we have to check the character
{ ch = fgetc(fp);                  // is end of the file character
putchar(ch);                       // or not, for this we have
                                   // a function feof() this
}                                  // means if the character is
}                                  // not the end of file character
                                   // it will return false ie. 0
                                   // and will not enter loop
                                   // so we have to use
                                   // logical not operator
```

// So that when the character read is not the end of character it should enter the loop and ~~print~~ read and print that characters one by one. When the character is end of file character it returns ~~true~~ so it should not enter the loop so when true it enters so !(1) = 0 now it will exit from the loop when it finds end of file character.

So in this way we can handle errs of files using ferror() and feof() functions.

perror()

It is a standard library function which prints the error messages specified by the compiler while performing read operation or a write operation on a file.

- The function perror() stands for print error.
- In case of an error, the programmer can determine the type of error that has occurred using the perror() function
- When perror() is called, then it displays a message describing the most recent error that occured during a library function call or system call. Its prototype can be given as:

```
void perror (char *msg);
```

- The perror() takes one argument which points to an optional user-defined message. The message is printed first followed by a colon and the implementation-defined message that describes the most recent error.
- If a call to perror() is made when no error has actually occured then "No error" will be displayed.
- The most important thing to remember is that a call to perror() and nothing is done to deal with the error condition, then it is entirely up to the program to take action.
  For example, the program may prompt the user to do something such as terminate the program.

- Usually, the program's activities will be determined by checking the value of "errno" and the nature of the error.
- In order to use the external constant errno, you must include the header file ERRNO.H.

Program illustrating the use of perror(). Here assume the file "file.txt" does not exist.

```
#include <errno.h>
#include <stdio.h>
#include <stdlib.h>
int main()
{
    FILE *fp;
    rename("file.txt", "newfile.txt");   //rename if there is any file.
    fp=fopen("file.txt", "r");
    if(fp==NULL)
    {
        perror("Error:");   return(-1);
    }
    fclose(fp);
    return(0);
}
```

Op: Error: No such file or directory.

P.T.O

positioning functions

In C, positioning functions reposition the file pointer in a file.

They are:

a) fseek() function
b) rewind() function
c) ftell() function.

} These functions play a very important role in random access of a file.

fseek() function:

What is this function?
Why we use this function?
How to use this function?

What is fseek() function?

Suppose we have a file "sample.txt" file and in this file I have the contents.

sample.txt

hello students
welcome to
C programming
lectures.

fseek() function is going to move, the cursor or the file pointer to a specific position.

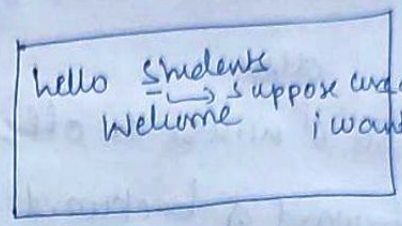
suppose i want to read only "C programming lectures" here. But when you open this file it will read what? the cursor will be in the begining., but i dont want to read this "hello students welcome to" and i want to print "C programming lectures" only.

so we have to move the cursor where "C" starts. and from that position till end of the file you can read (we can write down the while loop for reading).

→ for moving this cursor from starting to the character from which you want to read, we use fseek() function.
means we are moving a cursor from a specific position.

→ like rewind() function is used from moving the cursor from end to the beginning (or from any position to begining).

→ The cursor suppose is at S and it is not in the beginning, the cursor is to a specific position.



from there also we can move to any position for suppose i want to move it to second position i.e. 'e' of hello position.

→ from end to previous any other position also we can move.

→ with the help of fseek() we can do all these things.

→ we can also update any word with other word for suppose welcome with "Good morning". this also we can do.
we can update this file with fseek() function.

→ just we need to take care in which mode you are opening this file.
main thing is if you want to move the cursor from a specific position we use fseek() function.

for position the datatype is int.

syntax of fseek() function:

fseek (file pointer, offset, position/origin). → 3 arguments it contains.

obviously file pointer datatype is FILE. for offset the datatype is long int.
and the return type of fseek is 'int'. ⇒ this will return '0' if successfully, successfully it has updated the file pointer
otherwise it will return a non-zero value.

⇒ fileptr → obviously which file pointer is pointing to the file that we want to write.

⇒ offset → means how many bytes you want to move from a particular position.

"hello world"

Ex: from the beginning let us say the cursor is at "h" here I have to move to 'o' so 6 bytes we have to move (including space also.

so '6' will be the offset (how many bytes you want to move forward or backward.

⇒ If you want to move forward then we give positive value ⇒ +

⇒ If you want to move backward then we give negative value ⇒ −

⇒ position → means from which position you want to add the offset means you want to move the cursor from the beginning or from a particular position or from end, from which position you want to move. (you want to skip the content).

So this position is going to have 3 values like:

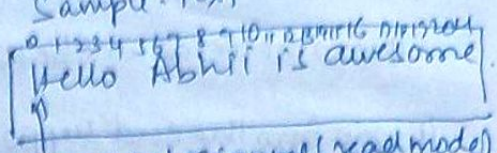
SEEK_SET ⇒ means the position is beginning of the file. (i want to move from beginning).

SEEK_CUR ⇒ means the current position of the file pointer.
(from the current position of the file pointer i want to move).
(backward or forward).

SEEK_END ⇒ means the position is End of the file
(i want to move from end of the file) backward.

How we will use these 3 positioning functions and how to use this fseek function in the program

Sample.text



Cursor is in beginning (read mode).

```
#include<stdio.h>
void main()
{
    FILE *fp = NULL;
    char ch;
    fp = fopen("sample.txt", "r"); // opening the file in read mode.
// we can use any mode but for simple understanding we are using
// r mode. and in every mode it will show different results.

    if(fp == NULL)
    {
        printf("Can't open file");
        exit(1);
    }
// supose i want to read from Abhii is awesome, so first i have
// to move the cursor from 0th index to 6th index (position) so how we
// will write this.
    fseek(fp, 6, SEEK_SET);   0+6=6 (cursor now will be at 6 index.)
                              ↳ from beginning position i am moving.
                6 bytes i want to skip and move the cursor to that position.
                (offset)
    ch = fgetc(fp);  // single character i want to read
    printf("%c", ch); // it will print A and if you want
// After printing A, the cursor  // to write something in this position
moves to 'b'. the file points to  // open the file in 'w' mode.
7th index now.
// suppose now i want to move 3 positions backward. ie. i want to
// print 0. - so how you can move -> simply you can write down.
        as follows.
```

fseek(fp, -3, SEEK_CUR); // we are moving backward from the current position (7-3=4) at 4th index.
ch = fgetc(ch);
printf("%c", ch); // It prints o value.

// Now the cursor will be at 5th index.

// Now from the end suppose i want to print o of (Awesome) Hi
// from current position also we can move, from beginning also
// we can move, but i want to move from end.

fseek(fp, -3, SEEK_END);
ch = fgetc(ch); // prints o
printf("%c", ch);
}.

// so if you want to move the cursor from a specific position, you can use fseek and these seek-set, cur, End are the three variables you can use.

This is a simple example of using fseek function using open a file using read mode.

// And after moving the cursor to a specific position we can update it.

If you are opening in r+ mode → it is also for reading and writing you can update here.

But if you are opening in w+ mode, it will the previous content will get erased so here it will create same problem.

from beginning you are moving your offset to 6th position, nothing will be printed. because once you open in w+ mode, the previous content gets erased. So it prints nothing. but in r+ mode it can print because the content will be as it is.

ftell() function():

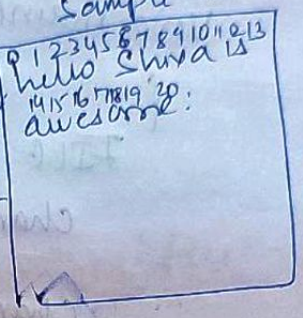
While we are randomly accessing the files we use ftell() function.

This function has its own specific applications.

What is ftell() function inc:

As the name suggests it will tell something, so what it will tell-?

It will tell the position of the file pointer in that file with respect to starting of the file.



At starting if you open the file in read mode, then the file pointer will be at the starting index i.e. 0

this is starting of the file.

so the position of this file pointer is 0th index.

So → using ftell function, you can find out the position of the file pointer.

↳ if suppose you have moved this file pointer to a specific position using a "fseek" function, sup pose i have moved the fp to 'S' and after this if you use ftell function it will return 6, because the index of this 'S' is 6. from the starting, you have skipped 6 bytes and at the 7th byte the file pointer is right now.

ftell → will tell the position of that file pointer. Whether it is pointing to starting of the file not end of the file.

→ one application of this ftell function is used to find out the length of this file.

↳ Now how you will find out the length?

Program:

```
#include<stdio.h>
void main()
{
    FILE *fp=NULL;
    char ch; int position;
    char str[50];
    fp=fopen("sample.txt","r");
    if(fp==NULL)
    { printf("can't open file");
      exit(1);
    }
// Now at last we will print where is the file pointer. For
// this we directly use ftell function.
position=ftell(fp); // ftell accepts only one argument
                    // that is the file pointer.
                    // and return type of ftell fun.
                    // is what?
                    // long int ftell(FILE *)
                    // o/p: Pos=0 or position
                    // of named pointer
```

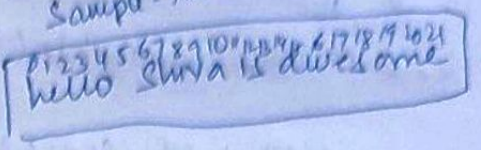
So it returns integer value.

printf("%d", position);

fclose(); }

Programs print the string:

sample.txt




```
#include<stdio.h>
void main()
{
    FILE *fp;
    char ch; int position.
    char str[50];
fp=fopen("sample.txt", "r");
if(fp==NULL)
  { printf("cant open file");
    exit(1);
  }
position= ftell(fp);
printf("%d", position); // 0
// Now suppose i will move the file pointer using fseek.
    fseek(fp, 5, SEEK_SET); // in this case file pointer
// moves to 5th index and here we have space.
    printf("%d", ftell(fp)); // gives 5
// fseek(fp, -3, SEEK_CUR);
// printf("%d", ftell(fp)); // returns 3
    ch= fgetc(fp);
    printf("%d", ch); -> prints space and fp moves to
                         next one.
    printf("%d", ftell(fp)); //6.
// if here you want to print the Complete string using
                fscanf also you can do.
Now the fp is pointing to 5.(6 index) and thus
                Shiva will be considered as
                a string not space
```

fscanf(fp, "%s", str); // Shiva will be stored in str.
↓ printf("%s", str); // prints Shiva.
used to read from the file (formatted read).

printf("%d", ftell(fp)); // It will print 11
}.

Note: If space occurs it is treated as end of the string.

Program to find length of the string using ftell() function.

How to find out the length of the complete file?

```
#include<stdio.h>
void main()
{
    FILE *fp = NULL;
    char ch; int pos;
    char str[20];
    fp = fopen("sample.txt", "r");
    if(fp==NULL)
    { printf("Can't open file");
      exit();
    }.
fseek(fp, 0, SEEK_END); // we are going to move
```

the file pointer backward (offset is 0 only) so file pointer will be pointing to the end character only.

// Now we can simply print the position of the file pointer and whatever is the position that will be the length of the file.

```
printf("%d", ftell(fp));
fclose();
}
```

This is how we will find out the length of the string using ftell.

3. rewind() function:

What is rewind() function.
Why we use this
How we use this in the program

What is rewind() function

rewind() means basically, as the name suggests, we are rewinding something, means, in a file suppose we have filename is demo.txt and in this file we have "hello students :C is great programming language this" text we have in the file.

→ Now at some point of time, my file pointer is pointing to last or maybe at last of students, but I want to read the complete string i.e. I want to set the file pointer to the beginning where ever "fp" may be, for this we use rewind() function.

→ Just have to use:

```
rewind(fp); // It is not going to return anything.
            return type is void.
```

↳ only one argument we need to pass

Where ever the fp is pointing to when we call rewind function, that file pointer will be shifting to the beginning of the file.

program:

```
#include<stdio.h>
void main()
{
  FILE *fp;
  char ch;
fp=fopen("demo.txt","r+");  // cursor points to beginning of the file
  if(fp==NULL)
  {
    printf("error");
    exit(1);
  }
// If you want to read the contents of the file write the code
// as below.
while(!feof(fp))
  {
    ch=fgetc(fp);
    putchar(ch);
  }
// But if i have moved the cursor to some position
fseek(fp,6,SEEK_SET);  // now the cursor moves to 6th index.
                       // At 0+6 offset we are
                       // starting+offset = 6th index
                       // so now the cursor points to the 6th index
this complete loop is written again.
  fclose(fp);
}
```

demo.txt

hello students C is great programming language

you can read the contents of file till complete end of file using this while loop. if the text is in a single line.

o/p: students C is great programming language.
Not hello.

but suppose after this I want to print from starting only the complete string i.e "hello --- soon" the complete text.

So we have to move the cursor to beginning so after the while loop and before fclose, simply write rewind(fp); then if you again write this reading while loop o/p will be the complete string.

program:

```
#include<stdio.h>
main()
{ FILE *fp;
  char ch;
 fp=fopen("demo.txt","r+");
  if (fp==NULL)
  { printf("error");
    exit(1);
  }
 fseek(fp,6,SEEK_SET);
while(!feof(fp))
  { ch=fgetc(fp);
  } putchar(ch);

 rewind(fp);
 while(!feof(fp))
   { ch=fgetc(fp);
     putchar(ch);
   }
}.
```

o/p: students c is great programming language.
hello students c is great programming language.

Try the programs with different modes.