eg →
```
#include <stdio.h>
enum color
{
   red=10;
   green;
   blue;
};
int main()
{
enum curr_color = red; // int curr_color = red; → this can also be used.
Pf("Red val= %d\n", red);
curr_color = green;
Pf("Green val= %d\n", green);
curr_color = blue;
Pf("Blue val= %d\n", blue);
}
```

O/P :- Red val = 10
Green val = 11
Blue val = 12

Note → By default enum initialized with '0'
⇒ red ++ ✗ not possible bec. red is constant

Session - 87

FILE

1-10-20

Files
→ File is a collection of data/information which is present in the secondary memory (Hard disk)
→ Holds permanent data.

File handling using C
⇒ RAM → primary memory → does not store data permanently
⇒ HARD DISK → secondary memory → store data permanently
→ A named collection of data stored in secondary memory is called File.

Need of file:-

→ whenever a program is terminated, the entire data is lost. Storing in a file will preserve data, even if the program is terminated.

Library functions to work with files are:-

1. fopen()
2. fclose()
3. fgetc()
4. fputc()
5. fgets()
6. fputs()
7. fprintf()
8. fscanf()
9. fread()
10. fwrite()
11. ftell()
12. fseek()
13. fclose
14. rewind()
15. remove()

Handling operations:-

1. Opening the file.
2. Read/write data operations on files
3. Close file

eg fopen() → opening the file
fclose() → closing the file

fopen:-

Function Prototype

```
FILE *fopen(const char *path, const char *mode);
```

description:-

→ fopen() opens the file whose name is the string pointed by path and associated a stream with it.

(passing address → catch with ptr)
fopen → returns one file ptr
(passing address of ptr → catch with double ptr).

Return value:-

→ fopen → opens the file
→ upon success, it will return file
→ upon failure, it will return null or zero '0'

eg:-

```
#include<stdio.h>
void main()
{
//char *cp;
FILE *fp;
fp=fopen("data","r");
if(fp==0)
{
Pf("File not present\n");
return;
}
else
Pf("File present\n");
}
```

FILE → typedef datatype

name of the string
the file is in "read mode"
Here, 'data' is the name of the file.

eg:-2 During run time

```
#include<stdio.h>
void main()
{
FILE *fp;
char s[20];
Pf("Enter the file name\n");
Sf("%s",s);
fp=fopen(s,"r");
if(fp==0)
{
Pf("File not present\n");
return;
}
else Pf("File present\n");
}
```

← the file is in read mode

i/p on load time: command line arguments

```
#include<stdio.h>
void main(int a, char **b)
{
  FILE *fp;
 fp= fopen(b[1], "r");
 if(a!=2)
 {printf("usage:./a.out filename\n");
  return;
 }
 if(fp==0)
 printf("file not present\n");
 else
 printf("File present\n");
}
```

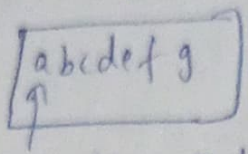
→ Instead of giving the filename alone, we can provide either absolute path or reference path of the file.

Modes:-

No.	Modes	Text file opens for	Creates new file	File ptr position	Truncate data
1	r	Reading	No	Begning of file	No
2	r+	reading/writing	No	Begning of file	No
3	w	Writing	Yes	Begning of file	Yes
4	w+	writing/reading	Yes	Begning of file	Yes
5.	a	Appending	Yes	End of file	No
6	a+	Reading/ appending	Yes	Begning of file for reading & end for appending	No

⇒ "r+"

→ overwriting happens when new data is written



eg:- fopen("abc", "r+");

if the file abc is present, the pointer is placed at the begning to read & write.

if the file is not there, it return '0'

⇒ "w"

→ If the file is not present, the file is created

→ If the file is present, it is truncated.

⇒ "w+"

fopen("abc", "w+");

old data is lost (truncated) and new data is written and saved.

⇒ "a" In append mode only writing is possible

⇒ "a+" → Reading is done from begning of appending/writing is done at end.

→ Create new file, if the file is not present

Absolute path → always starts from home

Reference path → start from .. or .

.. → Parent directory

. → Present working directory

Session - 88

21-10-20

User defined cp command (copy)

Generally, in pre-defined copy command

cp source destination

cp command opens source file in read mode and opens dest file in write mode.
If the dest file is not present, it will create the dest file.

`int fgetc(FILE * stream);`

The file ptr gets incremented automatically

Description:-

fgetc() fetches the data char by char in a file

Upon succes → returns the char as unsigned char, cast to integer

Upon failure → returns '-1' or EOF

End of file

eg I

Program to read the contents in the file char by char

This program functions as cat command

```
#include <stdio.h>
Void main(int argc, char **argv)
{
    char ch;
    if(argc != 2)
    {
      Pf(" usage :- /a.out fname \n");
      return;
    }
  FILE *fp;
   fp = fopen(argv[1], "r");
   if (fp == 0)
   {
      Pf("File not present \n");
        return;
   }
```

```
while((ch=fgetc(fp))!=-1)
  pf("%c",ch);
}
```

eg 2

```
#include<stdio.h>
void main(int argc, char **argv)
{
 char ch;
 if(argc!=3)
 {
  pf("usage:-/my_cp sf df\n");
  return;
 }
FILE *fs, *fd;
 fs=fopen(argv[1],"r");
 if(fs==0)
 {
  pf("source file not present\n");
  return;
 }
 fd=fopen(argv[2],"w");
 while((ch=fgetc(fs))!=-1)
 {
  fputc(ch,fd);
 }
}
```

the ptrs are incremented automatically

char by char is get from the src file & has to be put char by char in dest file

while compilation,

cc my_cp.c -o my.cp ↵

Note:- No need to close the file explicitly, the file is closed automatically

eg 3

```
#include<stdio.h>
void main(int argc, char **argv)
{
 char ch, of;
 if(argc!=3)
```

~~multiple cp~~ command

```
{
    Pf("usage:./my_cp sf df\n");
    return;
}
FILE *fs, *fd;
fs = fopen(argv[1], "r");
if(fs==0)
{
    Pf("Src file not present\n");
    return;
}
fd = fopen(argv[2], "r");
if (fd)
{
    Pf("The dest file already present\n");
    Pf("If you want to truncate, press y\n");
    Sf("%c", &op);
}
if(op=='y'||op=='Y')
{
    fd = fopen(argv[2], "w");
    while ((ch=fgetc(fs))!=-1)
    fputc(ch, fd);
}
```

If dest file is present use to overwrite it as per user wish

Q:)

```
#include<stdio.h>
void main(int argc, char **argv)
{
    char ch, op;
    if(argc!=3)
    {
        Pf("usage:./my_cp sf df\n");
        return;
    }
FILE *fs, *fd;
fs = fopen(argv[1], "r");
if(fs==0)
{
```

```
        Pf("Source file not present \n");
        return;
    }
fd = fopen(argv[2], "r");
if(fd)
{
    Pf("The df file is already present \n");
    Pf("If you want to truncate it press y\n");
    Sf("%c", &op);
}
else
goto L1;
```

instead of goto, we can give just

```
else
op = 'y';
```

```
    if(op == 'y' || op == 'Y')
    {
L1  : fd = fopen(argv[2], "w");
    while((ch = fgetc(fs)) != -1)
        fputc(ch, fd);
    }
}
```

Session - 89

3-10-20

Copy one source file to multiple dest. files.

eg:- ./a.out Sf d1 d2 d3 d4

0 1 2 3 4 5

Here, one ptr to source file & one ptr to dest. file is enough.

rewind(fs) ← src file ptr is again moved to the beginning.

eg 1:

```
#include<stdio.h>
void main(int argc, char *argv[])
{
    char ch;
    int i;
if(argc < 3)
{
    Pf("usage: ./a.out Sf d1 d2 d3 \n");
} return;
```

```
FILE *fs, *fd;
    fs= fopen (argv[1], "r");
    if(fs==0)
    {
      Pf("Src file not present ...\n");
      return;
    }
  for(i=2; i<argc; i++)
  {
    fd= fopen(argv[i], "w");
    while((ch=fgetc(fs))!= EOF)
      fputc(ch, fd);
    rewind(fs);
  // fs=fopen(argv[1], "r");// This line can also be written
  }                          Instead of rewind();
}                            If rewind is not used, the file ptr
                             points to the end only.
                             using this fun. it is shifted to
                             begining.
```

Finding the size of file.

$ ls -l data → gives the size of the file in the form of bytes

(cmd line argument)

Even the '\n' are considered while counting the size of file.

eg 2:

```
#include<stdio.h>
void main(int argc, char **argv)
{
  int i;
  char ch;
FILE *fp;
if(argc!=2)
{
  Pf("usage: ./a.out fname\n");
  return;
}
```

```
fp= fopen(argv[1], "r");
    if (fp==0)
    {
      pf("File not present \n");
      return;
    }
      c=0;
      while ((ch= fgetc(fp))!=-1)
        c++;
    pf("c= %d \n",c);  // gives the size of file in bytes
}
```

Finding how may times given char is present in given file

eg: `./a.out fname char`

eg ./a.out data a

→ open the file in read mode.
→ read the contents char by char
→ match it with the given char
→ then increment the count.

```
#include <stdio.h>
void main (int argc, char **argv)
{
  int i;
  char ch;
  FILE *fp;
  if (argc!=3)
  {
    pf("usage: ./a.out fname char \n");
    return;
  }
  fp= fopen(argv[1], "r");
    if (fp==0)
  { pf("File not present \n");
```

argv[2]= gives the address
argv[2][0]= gives the first char in address

```
    }
    c=0;
while((ch=fgetc(fp))!=EOF)
{
  if(ch==argv[2][0])
    c++;
}
printf("Count = %d\n",c);
}
```

Replacing one char with another char in the file.

argv[2][0] argv[3][0]

eg:- ./a.out data a m ← replacement char

↓ fname ↓ char to be replaced

Eg:-

```
#include<stdio.h>
void main(int argc, char *argv)
{
  FILE *fp;
  char ch, *p;
  if(argc!=4)
  {
    printf("Usage:-./a.out fname char char\n");
    printf("Ex:-./a.out data a m\n");
    return;
  }
  fp=fopen(argv[1],"r+");
  if(fp==0)
  {
    printf("File not present\n");
    return;
  }
  while((ch=fgetc(fp))!=-1)
  {
    if(ch==argv[2][0])
      fputc(argv[3][0],fp);
  }
}
```

[Here we expected the same char to be replaced but the adjacent char is replaced]

After fetching the char, the file pointer is incremented. at this ptr, the file ptr is incremented to the next position, so the next letter replaced.

This program replaces the char adjacent to the given char because, the file pointer gets incremented, once it finds the char specified.

To overcome this issue, we follow the below steps.

① Find file size
② Allocating DMA
③ Array based replacement
④ Again copying into file

⇒
```
#include<stdio.h>
#include<stdlib.h>
void main(int argc, char *argv[])
{
  FILE *fp;
  char ch, *p;
  int i, c;
  if (argc != 4)
  {
    Pf("usage: ./a.out fname char char\n");
    Pf("Ex.. ./a.out data a m\n");
    return;
  }
  fp = fopen(argv[1], "r+");
  if (fp==0)
  {
    Pf("File not present\n");
    return;
  }
// ① Finding file size
  c=0;
  while((ch=fgetc(fp))!=-1)
  c++; ← once the size of the file is counted, the file ptr is pointed at the end of the file.
// ② Allocating DMA
  rewind(fp); ← moving the file ptr to the beginning of the file.
  p=malloc(c+1);
  i=0;
```

//③ Copying file content into an array

```
while ((ch = fgetc(fp)) != -1)
    p[i++] = ch;
    p[i] = '\0';
    rewind (fp);
```

//④ Array based replacement

```
for (i=0; p[i]; i++)
{
    if (p[i] == argv [2] [0])
        p[i] = argv [3] [0];
}
```

//⑤ Again copying into the file (array to file)

```
    for (i=0; p[i]; i++)
        fputc(p[i], fp);
}
```

Session - 90 05-10-20

fputs

Prototype

```
fputs (const char *s, FILE *stream);
```

Return value
Success:- Returns a non -ve no.
Failure:- EOF

eg:- fputs(p, fp);

It copies the data without the null byte ('\0');

fgets()

prototype

Array base add. — size — fp (file ptr)

```
char *fgets(char *s, int size, FILE *stream);
```

if (10) read only (9)

eg:- char s[20], *p;

p = fgets(s, 10, fp); → it gets 10-1 ⇒ 9 char

here if a new line, if a '\n' is encountered with 9 char, the reading is stopped

→ fgets() reads almost one less than size char from stream & stores them into the buffer pointed by s.

→ fgets() stops reading after an EOF or when a newline is encountered.

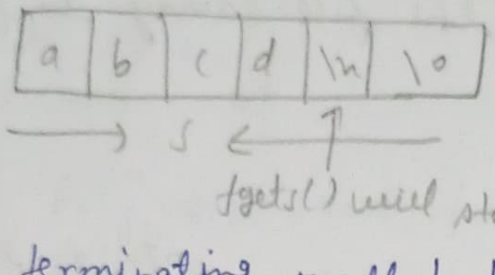
fgets() will read from where the file ptr is pointing to

→ If a newline is read, it is stored into buffer.

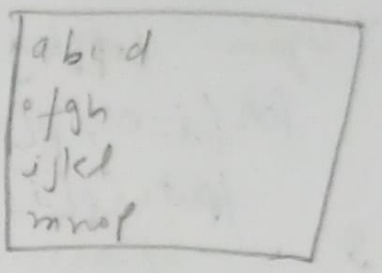
⇒ fgets() will read only one line

The fn can be used to read line by line

'\0' is put by the fgets(), as it encounted a '\n'.



file



→ A terminating null byte '\0' is stored after the last char in the buffer.

Return value:

success :- string base address

failure :- '0' (Zero)

eg:-

```
void main()
{
  FILE *fp;
  char s[20], *p;
  fp = fopen("data", "r");
  p = fgets(s, 10, fp);
  printf("p = %c  s = %s\n", p, s);
}
```

p = give the same string as *p

To read more line

```
while (p = fgets(s, 10, fp))
printf("p = %s  s = %s", p, s);
```

grep command

$ grep string filename
$ grep -n string filename
(-n: print the line no also)

→ grep command will print the entire line in the file, where the given string is present. → file

eg: grep abcd data

o/p:- abcd
adcdefgh

data:
abcd
abcdefgh
ijkl
mnop

→ grep command prints lines matching a pattern

Implementing user defined grep command using command line argument.

strstr() will return the base address where the sub-string is present

```
#include<stdio.h>
#include<string.h>
void main(int argc, char *argv[])
{
  char s[100];
  FILE *fp;
  if(argc!=3)
  {
    pf("usage:./~~a.out~~ my-grep string fname \n");
    pf("ex:-./my-grep hello data \n");
    return;
  }
  fp=fopen(argv[2],"r");
  if(fp==0)
  {
    pf("File not present \n");
    return;
  }
  while(fgets(s,100,fp))
  {
    if(strstr(s,argv[1]))
    pf("%s",s);
  }
}
```

this will continue & the fp will point to the next line string address

fgets return value acts as a condition for for loop.
fgets() upon success return address s will be main string. if there return but valid

making the while loop generic

① Find total file size
eg: 100 byte
s = malloc ((+1)

② while (fgets (s, c, fp) → total file size is mentioned.
s is overwritten everytime.
→ To point line no also
c=0;
while (fgets (s, 100, fp))
{
c++;
if (strstr (s, argv[1])
pf("%d %s", c, s);
}

Session-9 06-10-20

fprintf()

Prototype:- int fprintf(FILE * stream, const * format,....);
Description:- writes to formatted o/p to the given o/p stream.
Return value:-
Success:- Return the no. of characters
Failure:- Negative value
eg:-1 Void main() ← reading integer from file
{
FILE *fp;
int i;
fp = fopen ("data", "r");
fscanf("fp, "%d", &i); ← scan data from the file
printf ("i=%d\n", i);
}
eg:-2 Void main() ← writes integer data into the file
{
int i = 123456;
FILE *fp;

```
    fprintf(fp, "%d", i);
}
```

Eg: // copy an array of integer into the file

```
#include <stdio.h>
void main()
{
    int a[5] = {10, 20, 30, 40, 50};
    int i;
    FILE *fp;
    fp = fopen("data", "w");
    for(i=0; i<5; i++)
    fprintf(fp, "%d", a[i]);
}
```

← prints the array of integers one by one onto the file

Eg: // scanning int by int from the file & stored into the array

```
void main()
{
    int a[5];
    FILE *fp;
    fp = fopen("data", "r");
    for(i=0; i<5; i++)
    fscanf(fp, "%d", &a[i]);
    for(i=0; i<5; i++)
    Pf("%d", a[i]);
}
```

fscanf() stops scanning whenever a space/newline is encountered. (Provide a space after every int for ① Reading ② proper scanning

Eg: When 10 20 30 40 50 is given without space like 10203040 50, the entire string will be considered as a single int.

fprintf() → converts any data type into string & puts in an array in a file.

Copying & reading a string in a file

// copy a string to a file.

```
Void main()
{
  char s[20]= "hello";
   FILE *fp;
fp= fopen("data","w");
fprintf(fp,"%s",s);
}
```

→ // Reading data from a file

```
Void main()
{
  FILE *fp;
  Char s[20];
  fp= fopen("data","r");
  fscanf(fp,"%s",s);
  Pf("s= %s\n",s);
}
```

fscanf() → to scan word by word

Counting the size of file

```
c=0;
While (fscanf(fp,"%s",s) != -1)
  if(strcmp(s, argv[1])==0)   <- '0' when equal
    c++;
Pf("c= %d\n",c);
```

Copying a structure into a file

```
#include<stdio.h>
Struct ST
{
  int rollno;
  char name[20];
```

```
    float marks;
  }ST;
void main()
{
  ST s1;
  FILE *ff;
  ff= fopen("data", "w");
  fprintf(ff,"%d %s %f\n", s1.rollno, s1.name, s1.marks);
  };
```

Note:- Scanning the struct data from a file.

(While fetching a data from a file, the proper order of data should be followed, else unexpected o/p is shown).

```
Struct ST                    # scanning of struct data from
{                              a file.
  int rollno;
  char name[20];
  float marks;
} ST;
void main()
{
  ST s1;
  FILE *ff;
ff = fopen("data", "r");
fscanf(ff, "%d %s %f", &s1.rollno, &s1.name, &s1.marks);
ff("%d %s %f\n", s1.rollno, s1.name, s1.marks);
}
```

Session 9? 07-10-20

fwrite()

eg:

```
int a[5] = {10, 20, 30, 40, 50};
fwrite( a, sizeof(a[0]), 5, fp); // fwrite(a, sizeof(a), 1, fp);
```

- a → array base address
- sizeof(a[0]) → size of one element
- 5 → no. of elements
- fp → file pointer

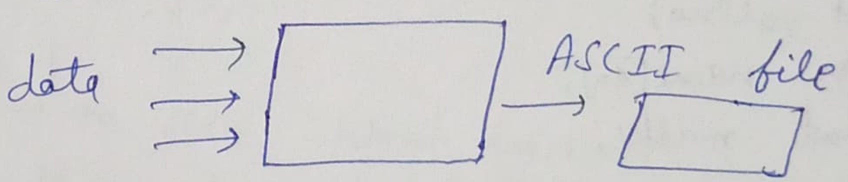
```
int i = 123456;
fwrite(&i, sizeof(i), 1, fp);
char s[20] = "hello";
fwrite (s, strlen(s), 1, fp);
```

Return value:-

Success :- Return the no. of items written

Failure :- 0

fprintf() will convert whatever data is present into ASCII.



fprintf

whatever the data we are storing into file is converted into ASCII.

eg: 1234567 → file size 7

fwrite()

Deals with binary. It writes as it is

eg: 1234567 (int) sizeof the file = 4 bec of int into 4 bytes

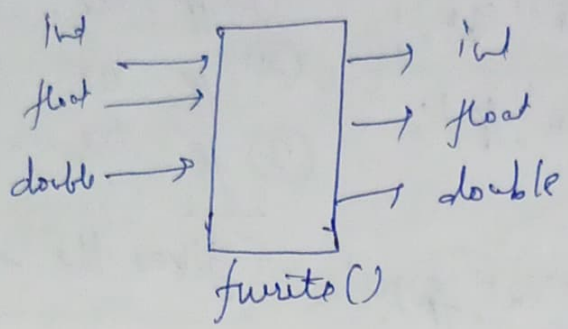
fread() is used to read the data whatever fwrite() has written

fprintf() → fscanf() → formatted data

fwrite() → fread() → unformatted data

↓

To use this we should aware of data present in file



eg:-

if data is large, with one time fun call, we can copy using fwrite().

eg: int a[5] = {10, 20, 30, 40, 50};

```
for(i=0; i<5; i++)
fprintf(fp, "%d", a[i]);
```

↓ called 5 times

```
fwrite(a, sizeof(a[0]), 5, fp);
```

↓ called only once

Advantage of fwrite()

1. Time (huge data in single function call)
2. Size is less (when there's huge data)

Prototype of fwrite()

```
fwrite(void *buffer, size_t length, size_t count, FILE *filename);
```

when dealing with string, fprintf() & fwrite() behave the same.

→ The o/p is readable.

→ The ASCII of the string is directly dumped.

Session - 9

08-10-20

```
void main()
{
  FILE *fp;
  char ch;
  fp = fopen("data","w");
  Pf("1) fp = %p\n", fp);
  ch = fgetc(fp);
  Pf("2) fp = %p\n", fp);
  ch = fgetc(fp);
  Pf("3) = %p\n", fp);
}
```

o/p :-

① fp = 0x----26

② fp = 0x----26

③ fp = 0x----26

gives the same address.

Note: All the header files are present in /usr/include

FILE ← is a typedef structure

```
typedef struct _IO_FILE FILE;
```

In this case,

FILE *fp;

where fp is a structure pointer.

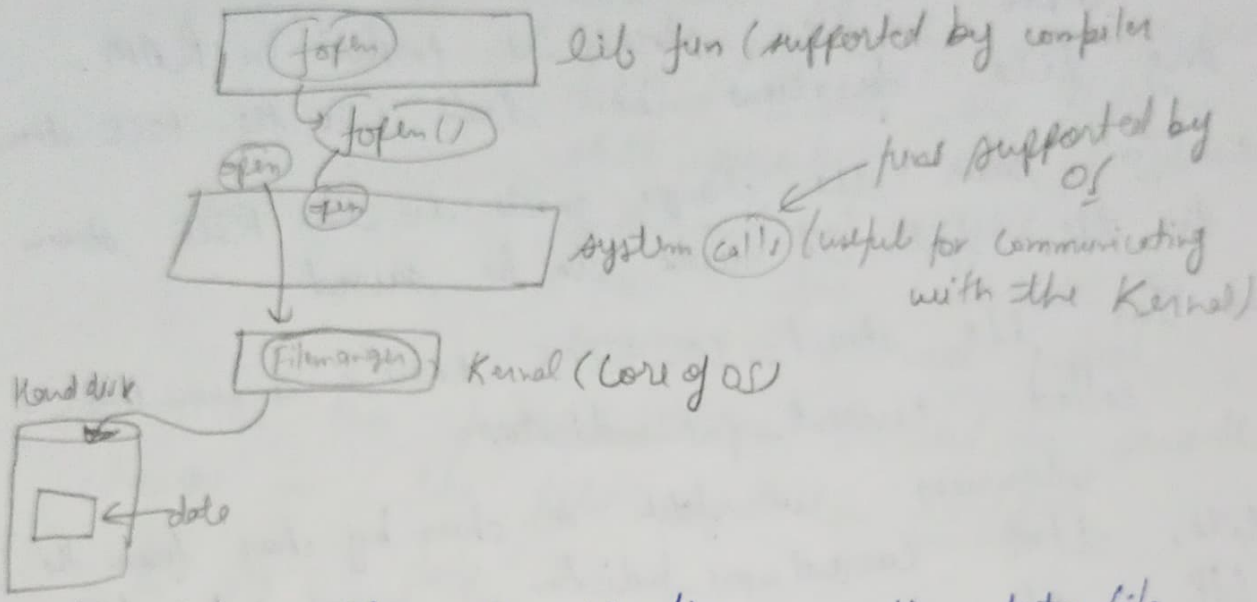
Pf("%ld %ld \n", sizeof(fp), sizeof(*fp));

o/p :- 8 - 216 ← size of struct FILE

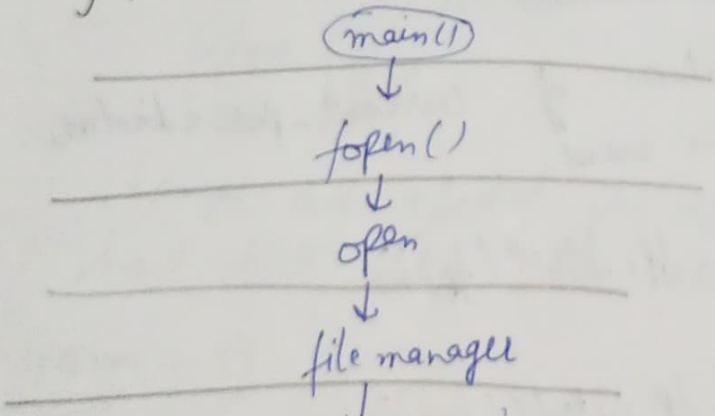
⇒ This depends on internal implementation (this is the result of internal implementation) result of ubuntu app on the top of windows OS. But it differ with actual ubuntu OS, & other apps such as Turbo C, Dev C++.

vi /usr/include/x86_64-linux-gnu/bits/types/FILE.h
(struct FILE is renamed) → gives the typedef of struct FILE

vi /usr/include/x86_64-linux-gnu/bits/libio.h → contains the declaration of struct FILE
(struct file is declared)



fopen() will not directly open the data file




```
fp = fopen("data", "r");
```

When fopen() is called, it allocates dynamic memory for _IO_FILE (fopen will give the addr. of the struct _IO_FILE)
(If the file is not present it returns null).
The file ptr (fp) = fopen("data", "r"), fp is pointing to the dynamically allocated structure & not the file. This is due to which, when a char by char is fetched, the ptr is not incremented. But actually, fp points

the structure base address.

→ fopen() after allocating memory, internally communicate with the file manager.

→ OS will allocate the FILE stream (a temporary buffer).

When a file stream is present in RAM. Any file operations is dealt with the FILE stream.

→ To save the changes made in the FILE stream to the file, it needs to be saved.

→ In the struct_IO_FILE there's a member called current_pos_indicator.

Whenever we fetch a char by char from the file, that current_pos_indicator is moved, but the file ptr is not incremented.

= To know the position of current_pos_indicator, ftell() function is used.

```
Pf("1) %p  %ld\n", fp, ftell(fp));
ch = fgetc(fp);
Pf("1) %p  %ld\n", fp, ftell(fp));
ch = fgetc(fp);
Pf("1) %p  %ld\n", fp, ftell(fp));
```

o/p:

```
0x----960   0
0x----960   1
0x----960   2
```

data type is long int

Similarly, when the rewind() is used, current_pos_indicator is moved to the beginning, not the file ptr. (When a file is opened, not the entire file is opened in RAM).