INTRODUCTION TO FILE HANDLING:-

In programming, when we involve large amount of data, the scanf() & printf() operations (I/O operations) get two major problem:-

(i) It is difficult to handle large volume of data through terminals.

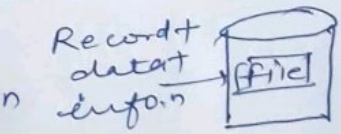
(ii) The entire data is lost when either the program is terminated or the computer is turned off.

One solution may be to take printout of the program & outputs. But this soln is not feasible when large amount of data is present.

Therefore it is necessary to have another solution where the data can be stored on disk. (Hard disk, compact disk) permanently. This method employs the concept of files to store data.

A file is a place on the disk where a group of related data is stored.

Now this new way of handling the data & infon is called file handling.



Need of file handling:- Due to the drawbacks of traditional I/P & O/P (I/O) system. there was a need of file handling

Drawbacks of Traditional I/O Systems :-

(1) until now, we were using console oriented. I/O functions.
Console application means an application that has a text based interface (black screen window).

(2) Most applications require a large amount of data. If this data is entered through console. then it will be a time consuming task.

(3) Main drawback of using traditional I/O :- data is temporary. (and it will not be available during execution)

So (1) Need way of dealing with data is file handling.
(2) data is stored onto the disk & can be retrived whenever required.
(3) output of the Prog. may be stored onto the disk.

FILE HANDLING IN "C"

File handling concept in c language is used for store data permanently in comp. using this concept we can store our data in secondary memory. All files related functions are available in stdio.h header file.

How to achieve File Handling in "C":-

(i) Naming a file.
(ii) opening a file.
(iii) Reading data from file.
(iv) Writing data into file.
(v) closing a file.

File opening Modes:-

S.No.	Mode	Meaning	Purpose
1.	r	Reading	open the file for reading only.
2.	w	Writing	open the file for writing only.
3.	a	Appending	open the file for appending (or adding) data to it.
4.	r+	Reading + writing	open for both reading & writing
5.	w+	writing + Reading	open for both reading & writing if the file exists, its contents are overwritten, if the file does not

exist. it will be created

6. a+ Reading + appending. opens for both reading & appending. if the file does not exist it will be created.

How the mode works!-

Mode functions

r - opens an existing file for reading only. gives error if the file doesn't exist.

w - Opens a new file for writing only. if the file does not exist then it will be created. else existing file will be destroyed & new file will be created.

a - opens an existing file for append purpose i.e. to add new data to it. if the file does not exist then it will be created

r+ - opens an existing file for reading & writing purpose. gives error if the file does not exist.

w+ - opens a new file for reading & writing if the file does not exist then it will be created., else existing file will be destroyed & new file will be created.

a+ - opens an existing file for reading & appending. if the file does not exist, then it will be created else existing file will be opened.

for opening text files use:-

r		rt
w		wt
a	OR	at
r+		r+t or rt+
w+		w+t or wt+
a+		a+t or at+

for opening binary files use -

rb
wb
ab
r+b or rb+
w+b or wb+
a+b or ab+

Functions used in file handling in C:-

① **fopen()** - Creates or opens a new file.

Syntax- `FILE *fp;`

↓ will be in upper case only.

Here fp is a pointer of file type.

To open a file:-

```
fp = fopen("file_name", "mode");
```

② **fclose()** - to close an existing file.

```
fclose(filepointer);
```

③ **getc()** - Read a character from a file.

④ **putc()** - Write a character in a file.

⑤ **fprintf()** - To write set of data in file. (formatted output)

⑥ **fscanf()** - To read set of data from file. (formatted input)

⑦ Syntax:-

③ `getc(fp);`
④ `putc(c,fp);`
⑤ `fprintf(fp, "control_string", list);`
⑥ `fscanf(fp, "control_string", list);`

(7) getw() - reads an integer from a file.
→ getw(fp);

(8) putw() - writes an Integer to a file.
→ putw (integer, fp);

(9) fseek() - set the position to desired point.

(10) ftell() - gives current position in the file.

(11) rewind() - Set the position to the beginning point.

(12) ~~fpti~~ fgets() - Read string of charaters from a file.

(13) fputs() - write string of characters to file.

(14) feof() - Detects end of file marker in a file.

Difference b/w Append and write mode:-

Write (w) mode & Append (a) mode, while opening a file are almost same. Both are used to write in a file. in both the modes. new file is created if it doesn't exists already.

The only difference they have is, when you open a file in the write mode, the file is reset. resulting in deletion of any data already present in the file. while in append mode this will not happen. Append mode is used to append or add data to the existing data

of file (if any). Hence when you open a file in append (a) mode., the cursor is positioned at the end of the present data in the file.

1) Program to open, close & write to a file :-

```
#include <stdio.h>
#include <conio.h>
void main()
{
    FILE *fp; char ch[20];
    fp = fopen("demo.txt", "w");
    fprintf(fp, "file handling prog.\n");
    fclose(fp);
}
```

filename

opened in write mode

string written to file pointed by fp.

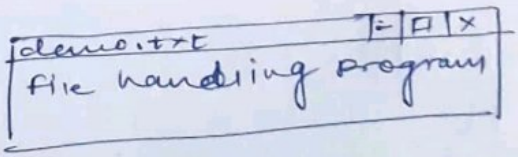
file closed.

→ fprintf(fp, "%s", ch);

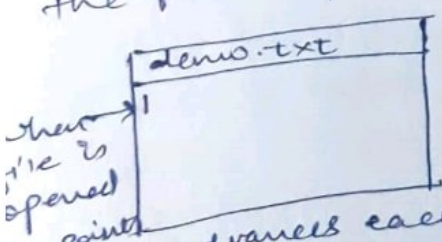
```
printf("enter the text:");
scanf("%s", &ch);
```

"%[^\n]" if we want to use spaces.

O/P → A text file with name demo.txt will be created in BIN folder of TC.

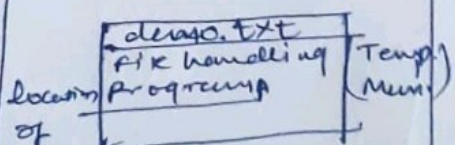


Steps :- ① Initially when the file is opened.

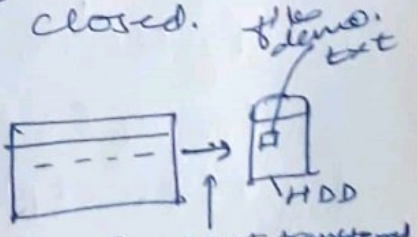


there file is opened fp points here. fp advances each no. of is written to file.

② After writing data to file.



③ When file is closed.



Content transferred to permanent storage

Reading from file:-

1. fscanf() - reads a string without space.
2. fgets() - reads a string with space.
3. fgetc() - reads a character from the file.

Prog. to read from file using getc():-

```
#include
main()
{
  FILE *fp;
  char c;
  fp = fopen("demo.txt", "r");
  do
  {
    c = getc(fp);
    printf("%c", c);
  }
  while (!feof(fp));
}
```

using fgets()-

```
#include
main()
{
  char str[50];
  FILE *fp;
  fp = fopen("demo.txt", "r");
  fgets(str, sizeof(str), fp);
  fprintf(stdout, "%s", str);
  fclose(fp);
}
```

← reading from file so. bytes of data & storing it in str.

↓ writing to screen using stdout

Appending data to a file:-

```
#--
#--
main()
{
    FILE *fp;
    char str[20];
    fp = fopen ("demo.txt", "a");
    fprintf (stdout, "enter the string to append ");
    gets (str);
    fprintf (fp, "%s", str);
    fclose (fp);
}
```

file opened in append mode
file pointer points at the end of the file

Checking the end of file:-

whenever a file is closed, a special character is placed at the end by the c compiler. This special character is known as end of file (eof) character. for checking the end of file, we use the function feof().

feof() → Returns zero if the end of file is not reached
feof() → Returns non-zero value if the end of file is reached.

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

```
ch = getc (fp);
while (feof (fp) == 0)
{
    putc (ch, stdout);
    ch = getc(fp);
}
fclose (fp);
}
```

→ As long as feof() returns zero, there is some data to read.