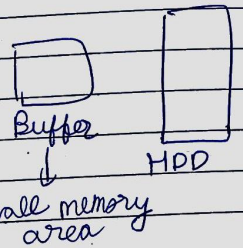
File - C supports files through which we can store data on files on disk and can retrieve later on in requirement

The file handling in C

- High level → managed by library func^n
- Low level. → managed by system calls.

Buffer



a small memory area

Buffer is a small memory where read/write tasks take place and when it is full it gets flush into HDD.

Steps for file operation :

1. Open a file
2. Read the data in file or write data into the file.
3. Closing the file.

fopen() - used for opening the file
fclose() - used for closing the file

We use a structure FILE in file handling. It is defined in stdio.h that contains all the information about the file like name, status, buffersize, current position, end of the file etc.

- in #include<stdio.h>
- Hidden from programmer

```
typedef struct {---
               ---
               ---
} FILE;
```

a file pointer is a pointer to the structure of type FILE, whenever a file is opened, a structure of FILE is associated with it

forex-

```
FILE *fp1, *fp2;
fp1 = fopen("myfile.txt", "w");
fp2 = fopen("yourfile.txt", "r");
```

"yourfile.txt" → file name; "r" → mode

Mode

"r" (read) → For reading the file the file to be opened must exist and

the previous data is not erased.

"w" → If file doesn't exist, a new file is created and if the file exist previous data get erased and new data is entered.

"r" (~~append~~) → same as "w" mode but we can also read and modify data.
w+ write + read
If file does not exist, a new file is created and if file exists previous data is erased.

"a" (append) - adding new contents at the end of file.

"r+" (read + write) → Same as r mode but in this mode, we can also write and modify existing data. The file to be opened must exist. This mode is also called update mode.

"a+" (append + read) → Same as 'a' mode. In this mode, we can also read the data. If file does not exists, new file is created.
If file exists, data is appended at the end of file. We can not modify existing data.

Structure of a general program:

```
main()
{
  FILE *fp;
  fp = fopen("filename.txt","mode");
  fclose(fp);
}
```

Formatted I/O-
fprintf = writes formatted data into a file

```
main()
{
  FILE *fp;
  char name[10];
  int age;
  fp=fopen("rec.txt", "w");
  printf("Enter name and age:");
  scanf("%s %d", &name, &age);
  fprintf(fp, "%s %d", name, age);
  fclose(fp);
}
```

fscanf() - Reads data from a file

```
#include <stdio.h>
struct student {
   char name[20];
   float marks;
 }stu;
main()
{
  FILE *fp;
```

```
fp = fopen("student.txt", "r");
printf("Name \t marks \n");
while (fscanf(fp, "%s %f", &stu_name, &stu_marks) != EOF)
printf("%s \t %f", stu_name, stu_marks);
fclose(fp);
}
```

Strings

1. No separate datatype for string
2. Array of type char
3. A character array is a string if it ends with a null character.
4. String constant or string literal :-
   "a" "Taj Mahal" "My age is %d and height is %f \n"

string constant.

- It is stored somewhere in the memory as an array of characters terminated by a null character ("\0").
- The string constant itself becomes a pointer to the first character of the array.

For ex- String "Taj Mahal" will be stored in memory as compiler adds null character automatically at the end.

1000	01	02	03	04	05	06	07	08	09
T	a	j		M	a	h	a	l	\0

compiler adds null character automatically at the end.

The string "Taj Mahal" is actually a pointer to the first character 'T'. So, whenever a string is used in the program, it is replaced by a pointer pointing to a string.

If we have a pointer variable like char *p, then we can assign the address of this constant to it as-

```
char *p = "Taj Mahal";
```

Program to show that identical string constants are stored separately

```
main()
{
printf("%u \n", "good");
printf("%u \n", "bad");
if ("bad" == "bad")
    printf("Same \n");
else
    printf("Not same \n");
}
```

exception: When the string constant is used as an initializer for a character array then it does not

PAGE:
DATE:

Pre The C Processor

Source code
↓
Preprocessor
↓
Compiler

Advantages of using preprocessor:

* It increases the readability of program.
* Program modification becomes easily.
* Makes the program portable and efficient
* The lines starting with # symbol are known as preprocessor directives.

Main functions performed by preprocessor directives:

1) Simple macro substitution
2) Macros with arguments
3) Conditional compilation
4) Including files
5) Error generation, pragmas and predefined
6) macro names

DEBUG

```
#include <stdio.h>
#define DEBUG
```

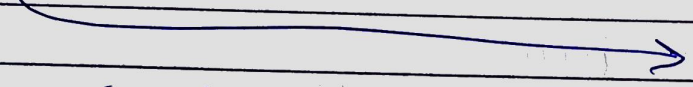
after macros

Macros with arguments -

Syntax

```
# define macro-name(arg1, arg2, ...)
    macro- expression
```

Ex-
```
# define SUM(X,Y)    ((X)+(Y))
# define PROD(X,Y)   ((X)*(Y))
```




```
main()
{
  int x;
  # if def DEBUG
      printf("Starting main\n");
  # endif
      .....
  func();

# if def DEBUG
  printf("Now values of X=%d and Y=%d",
          x, Y);
  # endif
  func() {
  # if def DEBUG
     printf("Inside function");
  # endif
  }
```

Some important function in string:

1) strlen() - counts the number of characters in string.

2) strcpy() - Ex - strcpy(s1, s2)

3) strcmp() - Ex - strcmp(s1, s2)

If =0 → equal

s1 > s2 → .

s1 < s2

4) strcat() = concatenate

#

#

```
main()
{
  char str1[20], str2[20]
printf("Enter the first string");
scanf("%s", str1);
printf("Enter the second string");
scanf("%s", str2);
strcat(str1, str2);
printf("First string: %s\t Second string: %s\n",
str1, str2);
strcat(str1, str2)
printf("First string: %s\t second string: %s
      \n", str1, str2);
strcat(str1, "_one");
printf("Now I string is %s", str1);
```

SPIRAL

PAGE
DATE

```
*
* *
* * *
* * * *
```

```
#include <stdio.h>
main()
{
  int i, j, n;
  printf("Enter n:");
  scanf("%d", &n);
  for(i=1; i<=n; i++) {
    for(j=1; j<=i; j++) {
      printf("*"); }
    printf("\n");
  }
}
```

```
for(i=1; i<=n; i++) {
  for(j=1; j<=i; j++) {
```