Unit 6 Files

Concept of Files :- What is Files in C

- Often data is so large that all of it can not be stored in memory and only limited amount of it can be displayed on screen
- m/m is volatile & its content would be lost once the program is terminated.
- Hence it is necessary to store that data in the file on the disk so that it can be later retrieved, used or displayed either part of it or whole.

Why files are needed?

- Data tobe displayed on screen (console) may be very large
- only limited amount of data can be displayed on console
- As m/m is volatile so it is difficult to retriere programatically generated ~~get~~ data
- If we need to do so we may need to store it on to the local file system which is volatile and can be accessed every time.
- Hence the concept of file handling is used in C.

File Handling in C (Operations performed on files)

- File handling in enable us to create, update read and delete the files stored on local file system through c pgm.
- Operations performed on files (File Operations in C)
  1. Creating new file
  2. Opening an existing file
  3. Reading from the file
  4. Writing to the file.
  5. Deleting the file
  6. moving to the specific location in a file(seeking)
  7. closing a file.

File Pointer in C

- A file pointer is a reference ~~to a~~ used to handle and manage files using standard I/O functions provided by C library.
- It is a pointer of type FILE which points to a structure containing information about a file such as name, status, current location in the file.
- It is used to keep track of location within the file & manage all file operation

Declaration of File Pointer:-

- To use file pointer it must be declared b4 its use as

```
FILE *fp;
```

of type

fp is a pointer ~~to a~~ FILE ~~object~~ structure
FILE structure is defined in std I/O Library (stdio.h)

- file pointer is a reference to a particular position in the opened file.
- It is used in file handling to perform all file operation such as read, write, close etc
- FILE is basically a data type and we need to create a pointer variable to work with it as *fp

Functions for File Handling

- There are many functions in C library used in file operation

1. fopen() – used to open an existing file or It creates new file & open it.
2. fprintf() – write data into the file.
3. fscanf() – reads data from the file
4. fputs() – writes a character into the file
5. fgets() – reads a character from file
6. fclose() – closes the file

7. fseek() - sets the file pointer to given position
8. ~~fputw() - writes an integer to file~~
9. ~~fgetw() - reads an integ~~
8. ftell() - returns current pos

Opening / Creating File :- fopen()

- file must be opened to perform any operation
- fopen() is used to create new file or open an existing file
- You need to specify the mode in which you want to open the file
- There are various modes ~~the~~ used to open the file.
- Any one of them can be used during creating / opening a file.
- fopen() takes two parameter.
  i) filename — name of file you want to open or create
  e.g. filename.txt
  ii) mode - It is a single character that represent in which mode you want to open a file (such as read write, append)
  w - writes to a file
  r - read from file
  a - appen new data to the end of file

~~WAP to create a file in write mode.~~

Syntax to create a file in write mode.

```
FILE *fp;
fp = fopen ("test.txt", "w");
fclose(fp);
```

- The file test.txt is created in the same directory as that of C's current-working directory
- If you want to create a file in a specific folder then provide absolute path

```
fp = fopen ("C:\\xyz\\test.txt", "w");
```

- fopen() is used to open a file & associate it ~~with~~ with file pointer using syntax

```
FILE *fp;
fp=open ("file_name", "mode");
```

file_name - name of file to be opened.

mode - specifies purpose of opening file such as (r: reading, w-writing, a: append etc)

How fopen() works:-

- fopen() is used to create/open file & associate it with file pointer
- This file pointer is used to perform various operation on file.
- fopen() is defined in stdio.h
- It returns pointer to the file structure
- This pointer keeps track of various details about file such as current pos & access mode.
- fopen() on successful execution returns the pointer of type FILE * pointing to file structure
- If fopen() is not successful then it returns NULL. If file can not be able to open due reasons like
  1. file does not exist (in r modes)
  2. Lack of permission to access the file
  3. Path or filename is incorrect.
- Steps of execution of fopen()
  1. First fopen() checks whether the specified file name & path are valid.
  2. Then its mode parameter is verified. If the mode is incorrect or unsupported then fopen() fails.
  3. If file already does not exist for 'w' or 'a' mode then fopen() creates new file

- If file is successfully opened then fopen() creates FILE structure, initilize its fields & return the pointer to it.
- Returns NULL on failure

WAP to implement fopen() to write in file

```
#include<stdio.h>
void main()
{
   FILE *fp;
   fp= fopen("sample.txt", "w");
   if (fp == NULL)
     {
       printf("Error can not open file");
     }
  fprintf(fp, "welcome to file");
  fclose(fp);
  printf("File created & data written
          successfully");
}
```

- It is always good practice to check if the file was opened successfully by verifying its return file pointer is NULL or not

fprintf()

- fprintf() used for formatted o/p to a file.
- It allow writing formated data to a file
- It allows writting formatted data to a file in a way similar to printf() used for printing on console
- This fun° is defined in stdio.h header file and is widely used for outputting text, numbers & other data types in file.
- Syntax:-

```
fprintf(fp, "_msg to_print|write");
        └ file pointer
```

Return Value:-

- fprintf() - It returns total no of characters written to file.
- On failure it return negative value (eg -1)

Format Specifier:-

- They are similar to printf()
  - %d - int
  - %f - float
  - %c - single char
  - %s - string

WAP to store student infomation in file

```
#include<stdio.h
void main()
{
    FILE *fp;
    char name[] = "xyz";
    int prn = 101;
    float marks = 90.90;
    fp = fopen("studeninfo.txt", "w");
    if (fp == NULL)
    {
        printf("Error: could not open file");
    }
    fprintf(fp, "Name = %s", name);
    fprintf(fp, "Prn = %d ", prn);
    fprintf(fp, "Marks = %f", marks);
    fclose(fp);
    printf("file operation is sucessful :)");
    fclose(fp);
}
```

* Reading from the file :- fscanf()

- fscanf() is used to read formatted i/p/data from a file
- It works similar to scanf() that read i/p from console
- fscanf() is used to read data from the file associated with FILE pointer.

WAP to Read & Write data into text file.

```
#include <stdio.h>
void main()
{
    int prn = 101;
    char name[30] = 'xyz';
    float marks = 90.90;
    fp = fopen ("studmarks.txt", "w");
    if (fp == NULL)
      printf(" Error in opening file");
    fprintf(fp, " PRN = %d ", prn);
    fprintf(fp, "Name = %s", name);
    fprintf (fp, "Marks = %f", marks);
```

```
#include<stdio.h>
void main()
{
  int i=10, a;
  float pi=3.1415 p;
  char c='Y', ch;
  FILE *p;
  fp = fopen ("file1.txt" , "w");
  fprintf ( fp , "%d %f %c " , i , pi , c);
  fclose (fp);
  fp = fopen ( " file1.txt " , "r");
  fscanf (fp, "%d %f %c", &a , &p , &ch);
  fclose(fp);
  printf ( " %d", a);
  printf ( " %f" , p);
  printf ( "%c", ch);
  getch();
}
```

Closing the File fclose() :-

- fclose() is a function used to close a file that was previously opened using fopen()
- closing a file is important because it ensures that all the data written to the file is properly saved, buffers are flushed and resources are released.

Syntax

fclose (FILE *stream).
└ file pointer

This file pointer is the fp that was returned by call to fopen()

i) on success it return 0

ii) failure it returns End of File char (i.e. -1)

Why fclose is used —

① Flushing data to disk — when you write to a file, the data might be stored temporarily in a buffer. The fclose() func flushes this buffer & ensure that all written data is saved to the disk.

② Freeing Resources — closing a file releases any resources that were allocated for the file

③ Preventing Data corruption - If file is not closed properly, data might not be written correctly leading to corruption.

fgets() :-

- fgets() is used to read content from the file up to the next line break and writes it into a char array
- A "\0" null terminating char is appended at the end of the contents.
- The position indicator (cursor) is moved to the next unread char in the file
- fgets() is defined in <stdio.h> header file.

- fgets() read (n-1) character and the last char is NULL char.
- It stops reading when it encounters newline (\n) char or reaches to end of file.
- After reading the string it append \0 to mark end of string.

fputs()

- It is a function defined in stdio.h
- It is used to write contents of file
- It returns 1 when write operation is successful else return 0
- fputs line writes single line of character in file.

WAP to write in the file using fputs(), fgets()

```
#include<stdio.h>
void main()
{
    char buffer[50];
    FILE *fp;
    fp = fopen("file2.txt", "w");
    fputs("Hello World", fp);
    fclose(fp);
    fp = fopen("file2.txt", "r");
    fgets(buffer, 50, fp);
    printf("Read Data : %s", buffer);
}
```

fseek()

- fseek() is used to move the file pointer associated with given file to specific position
- means it is used set the file pointer to the specified ~~location~~ offset
- It is used to write data into the file at desired location.
- Syntax

```
int fseek( FILE *stream, long int offset,
                                int position);
```

(stream: pointer)

pointer: - pointer to a FILE object that identifies stream

offset - It is the no. of bytes to offset from position.

position - It is the position from where the offset is added.

- It defines the point with respect to which the fp needs tobe moved
- It has 3 values

① SEEK_END - It denotes end of file.

② SEEK_SET - It denotes starting of file

③ SEEK_CUR - It denotes file pointers current position.

- It returns 0 if successful or else returns non-zero value.

e-g. fseek(fp, 0, SEEK_END); moving pointer to end.

fseek(fp, 4, SEEK_SET); moves to the begining of 4th line.