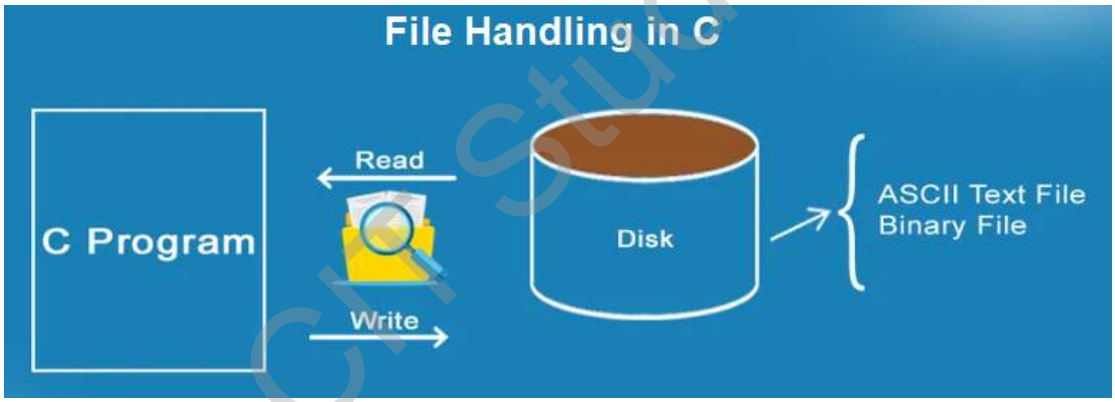
Part-1: Functions: Designing Structured Programs, Functions in C, User Defined Functions, InterFunction Communication, Standard Functions, Passing Array to Functions, Passing Pointers to Functions, Recursion

Part-2: Files I/O - Text Input / Output: Files, Streams, Standard Library Input / Output Functions, Formatting Input / Output Functions, Character Input / Output Functions

Files I/O - Binary Input / Output: Text versus Binary Streams, Standard Library, Functions for Files, Converting File Type.

File Handling in C

A file is a place where a program stores data permanently in a sequence of bytes.
The content available on a file is not volatile like the compiler memory in C. But the program can perform various file operations, such as creating, opening, reading a file, or manipulating the data present inside the file. This process is known as file handling in C.



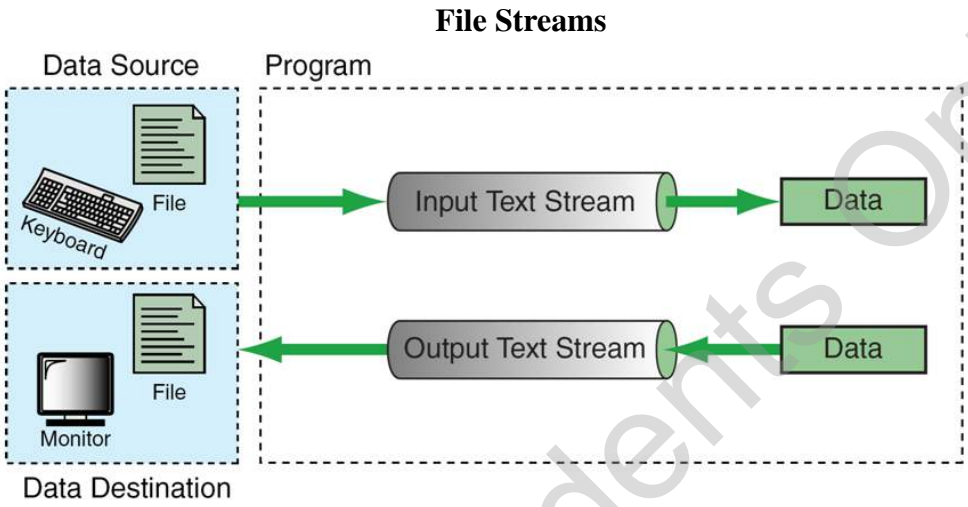
Need for file handling in C - The output generated by a program needs to be saved permanently to check or use at various times.

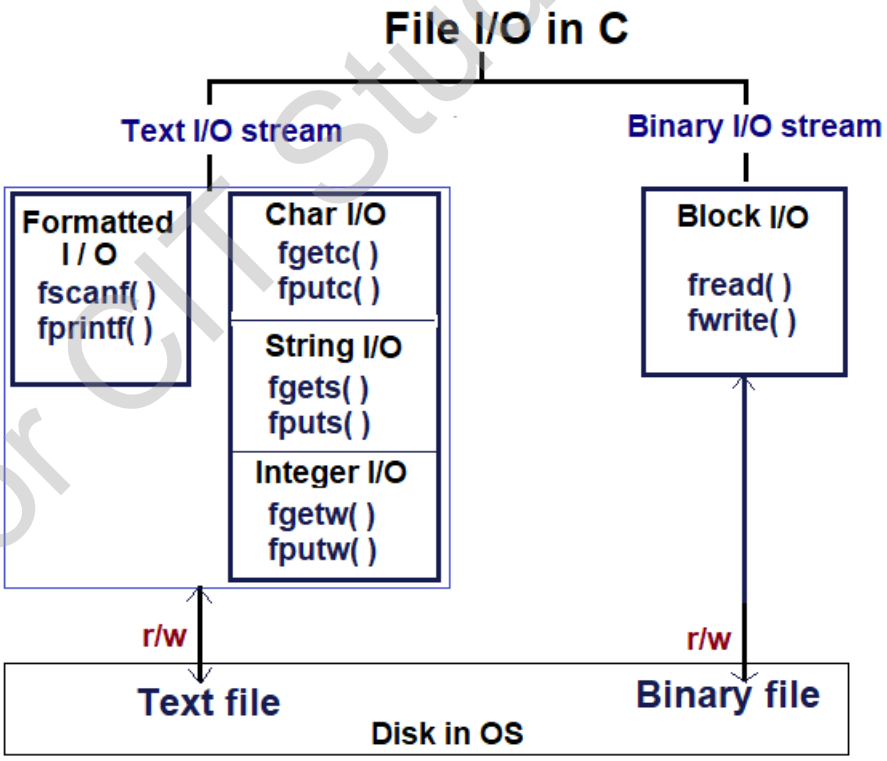
What are the types of file streams in C?
We have basically 2 types of file streams in C. They are namely Text I/O stream and Binary I/O stream.

1. **TEXT I/O stream -** The user can create TEXT Files easily using file handling in C. It stores information in the form of ASCII characters internally and when the file is opened, the content is readable by humans. It can be created by any text editor with a **.txt** or **.rtf** (rich

text) extension. Since text files are simple, they can be edited by any text editor like Microsoft Word, Notepad, Apple Text edit, etc.

2. **BINARY I/O stream-** It stores information as blocks or bytes in the form of **0 s** or **1 s** and it is saved with **.bin, .obj, .exe** extensions. Therefore it takes **less space**. Since it is stored in a binary number system format, it is **faster to access** and **more secure** than a text file.





What are the File Operations allowed in C?

The following is the list of file operations we can perform in the C programming language.

1. Creation of a new file (**fopen** with attributes as **"w" or "w+"** or **"a" or "a+"**)
2. Opening an existing file (**fopen**)
3. Reading from file (**fscanf or fgets or fgetc**)
4. Writing to a file (**fprintf or fputs or fputc**)
5. Moving to a specific location in a file (**fseek, rewind, ftell**)
6. Closing a file (**fclose**)
7. Removing a file (**remove**)

All the above operations are performed using the following file-handling functions available in C.

Standard Library Input Output Functions for Text & Binary Files in C

Function	Description	Syntax
File Open / Close		
fopen()	Used to create a new file or open the file if it already exists	FILE *fopen("file_name", "mode");
fclose()	Used to close existing file	fclose(FILE *fp)
Formatted I/O Functions in Files		
fprintf()	Used to write formatted data in file	fprintf(FILE *stream, "formatSpecifiers", variables)
fscanf()	Used to read formatted data from the file	fscanf(FILE *stream, "formatSpecifiers", &variables)
Character I/O Functions in Files		
fputc()	Used to write characters in a file	fputc(char ch, FILE *stream)
fgetc()	Used to read characters from a file	fgetc(FILE *stream)
String /O Functions in Files		
fputs()	Writes a string of characters into file (not including null \0 char).	int fputs(const char *s, FILE *stream)
fgets()	Reads a string of characters from file.	**char*** fgets(**char** *s, **int** n, **FILE** *stream)
Integral File I/O Functions		
fputw()	Used to write integral value in the file	fputw(int number,File *fp)
fgetw()	Used to read integral value from the file	fgetw(File *fp)

Functions to Move within a File		
fseek()	puts the file pointer to the specified place	fseek(FILE *stream, long int offset, int whence)
ftell()	It will return the current position of the file pointer in the file	ftell(FILE *stream)
rewind()	file pointer is set to the start of the file	rewind(FILE *stream)

Modes to Open a File in C

To open a Text or Binary file we use **fopen()** with different opening modes. The most common modes used are listed below.

Mode	Description	Example
File Modes for TEXT File Streams		
r	Opens a text file in read-only mode	fopen("myfile.txt", "r")
w	Opens a text file in the write-only mode	fopen("myfile.txt", "w")
a	Opens a text file in append mode	fopen("myfile.txt", "a")
r+	Opens a text file in both reading and writing mode.	fopen("myfile.txt", "r+")
w+	Opens a text file in both reading and writing mode. It sets the cursor position to the beginning of the file if the file already exists.	fopen("myfile.txt", "w+")
a+	Opens a text file in both reading and writing mode. The reading operation is performed from the beginning and the writing operation is performed at the end of the file.	fopen("myfile.txt", "a+")
File Modes for BINARY File Streams		
rb	Opens a binary file in read mode.	fopen("myfile.txt", "rb")
wb	Opens a binary file in write mode.	fopen("myfile.txt", "wb")
ab	Opens a binary file in append mode	fopen("myfile.txt", "ab")
rb+	Opens a binary file in read and write modes	fopen("myfile.txt", "rb+")
wb+	Opens a binary file in read and write modes	fopen("myfile.txt", "wb+")
ab+	Opens a binary file in read and write modes	fopen("myfile.txt", "ab+")

How to Create or Open a File?

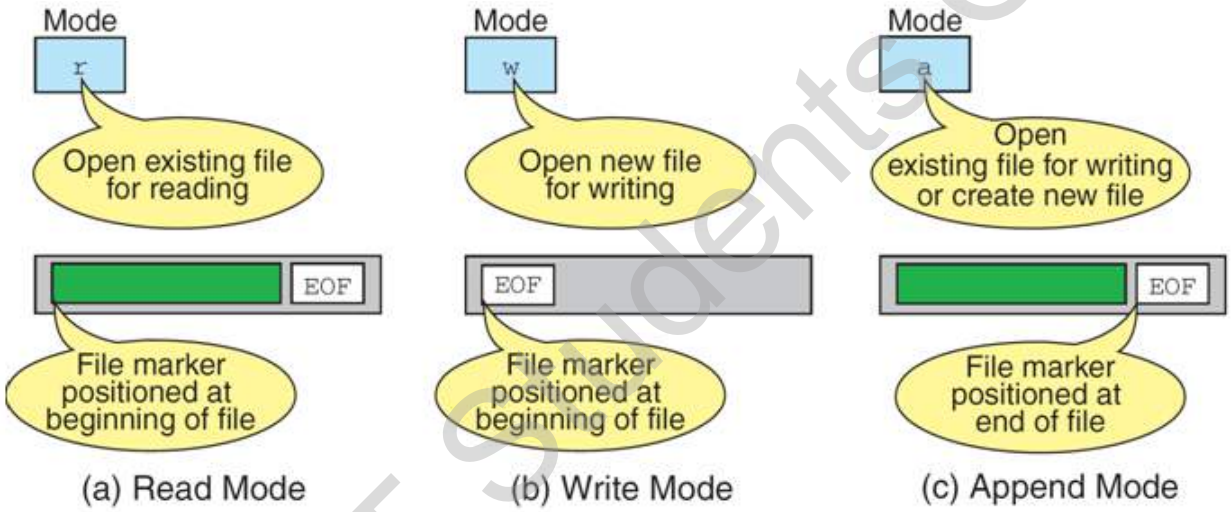
1. To create a new file or open an existing file, we need to create a file pointer of **FILE** type.
2. **fopen()** is a standard library function to create a new file or to open an existing file. A file can be opened in different modes such as read, write or append.

   Ex:

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

   Explanation:

   Creates a new file **students.txt** in writing mode if the file is already not available in OS.
   Opens the file **students.txt** in writing mode if the file already exists in the OS.



What are Formatted Input Output Functions in Text Files?

fprintf() - This function is used to **write** formatted data **to a file** (using format specifiers).
Returns EOF in case of error writing to file.

Syntax: fprintf(FILE *stream, const char *format [, argument, ...])

Ex:
```
fp = fopen("file1.txt", "w");
fprintf(fp, "Hi Rajasekhar, This text is written by fprintf. \n");
```

fscanf() - This function is used to **read** formatted data **from a file** (using format specifiers).
Returns EOF in case of error reading from file.

Syntax: fscanf(FILE *stream, const char *format [, argument, ...])

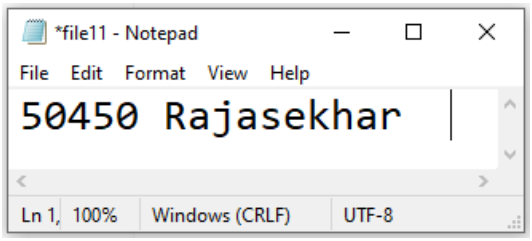
Ex:
```
fp = fopen("file1.txt", "r");
char buff[255];
fscanf(fp, "%s", buff)
```

Example: Program to write to and read from a text file using fprintf() and fscanf() functions

```
#include <stdio.h>
#include <stdlib.h>
int main()
{  int num=50450;
   char name[] = "Rajasekhar";
   FILE *fp;
   fp = fopen("C:\\myfiles\\file11.txt","w");
   if(fp == NULL)
   {  printf("File Error!");
      exit(1);
   }
   fprintf(fp,"%d %s",num, name); //writes num to file
   fclose(fp);
//To read num from file
   fp = fopen("C:\\myfiles\\file11.txt","r");
   if(fp == NULL)
   {  printf("Error! opening file");
      exit(1);
   }
   fscanf(fp,"%d %s", &num, name);
```

```
//screen output
   printf("\n Number = %d", num);
   printf("\n Name   = %s", name);
   fclose(fp);
   return 0;
}
```

Output Text File:



Output Screen:

Number = 50450
Name = Rajasekhar

What are Character Input Output Functions for Text Files?

The functions **fgetc()** and **fputc()** are the character input and output functions for text files in c.

fputc() - This function is used to write/insert a character to the specified file when the file is opened in writing mode.

Syntax: fputc(char, *file_pointer)

Ex:
```
fp = fopen("C:\\myfiles\\file3.txt","w");
fputc('A',fp);
```

fgetc() - This function is used to read a character from specified file which is opened in reading mode. It reads from the current position of the cursor. After reading the character the cursor will be at next character.

Syntax: fgetc(*file_pointer)

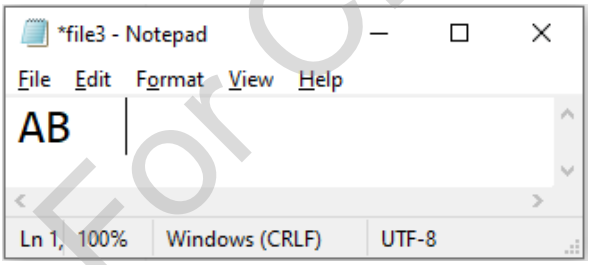
Ex:
```
char ch ;
ch = fgetc ( fp ) ;
```

Example: Program to write and read characters to and from a text file using fputc() and fgetc()

```
#include<stdio.h>
int main( )
{
FILE *fp ;
char ch ;
fp = fopen("C:\\myfiles\\file3.txt","w");
fputc('A',fp);
ch = 'B';
fputc(ch,fp);
fclose(fp);

fp = fopen("C:\\myfiles\\file3.txt","r") ;
while ( 1 )
{
    ch = fgetc ( fp ) ;
    if ( ch == EOF )
        break ;
    printf("%c",ch) ;
}
fclose(fp);
return 0;
}
```

Output Text File:



Output Screen:

AB

What are String Input Output Functions for Text Files?

The functions **fgets()** and **fputs()** are the string input and output functions for text files in c.

fputs() - This function is used to insert string data into specified file that is already opened in writing mode.

Syntax: fputs("string", *file_pointer)

Ex:
```
fp=fopen("file5.txt","w");
    fputs("Programming Through C by Dennis",fp);
```

fgets() - This function is used to read a string of characters from a file that is already opened. The reading starts from the current cursor position. The fgets() terminates its reading when it reaches a NULL character (\0) at the end of any string..

Syntax: fgets(stringVariableName, numberOfCharacters, *file_pointer)

Ex:
```
printf("%s",fgets(text,200,fp));
```

Example: Program to write and read string to and from a text file using fputs() and fgets()

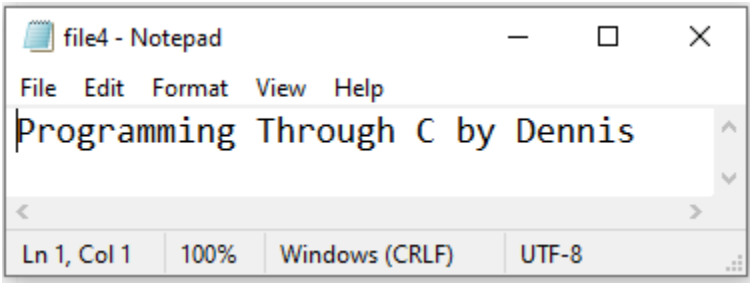
```
#include<stdio.h>
int main(){
FILE *fp;
char text[300];

fp=fopen("file4.txt","w");
fputs("Programming Through C by Dennis",fp);
fclose(fp);

fp=fopen("file4.txt","r");
printf("%s",fgets(text,200,fp));
fclose(fp);

return 0;
}
```

Output Text File:



Output Screen:

Programming Through C by Dennis

What are Block Input Output Functions for Binary Files?

The functions **fwrite()** and **fread()** are used to write and read bytes of data into and from binary files.

fwrite( ) - This function writes bytes of data into a binary file. This functions take four parameters:

1. address of data to be written in the disk
2. size of data to be written in the disk
3. number of such type of data
4. pointer to the file where you want to write.

Syntax: fwrite(addressData, sizeData, numberOfData, pointerToFile);

fread() - This function reads bytes of data from a binary file. The function fread() also take 4 arguments similar to the fwrite() function as above.

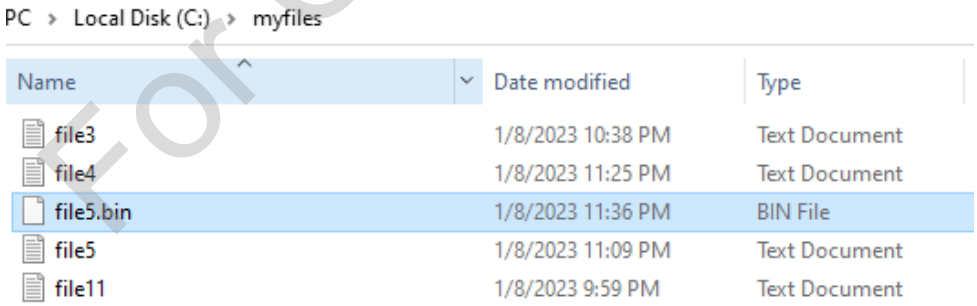
Syntax: fread(addressData, sizeData, numberOfData, pointerToFile);

Example: Program to write into and read from a binary file using fwrite() and fread()

```
#include <stdio.h>
#include <stdlib.h>
struct marks
{   int m1;
    int m2;
    int m3;
};
```

```c
int main()
{   struct marks std;
    FILE *fptr;
    fptr = fopen("C:\\myfiles\\file5.bin","wb");
    if(fptr == NULL)
    {
       printf("Error! opening file");
       exit(1);
    }
    std.m1 = 100;
    std.m2 = 50;
    std.m3 = 75;
    fwrite(&std, sizeof(struct marks), 1, fptr);
    fclose(fptr);
//To read from binary file
    fptr = fopen("C:\\myfiles\\file5.bin","rb");
    if(fptr == NULL)
    {   printf("Error! opening file");
       exit(1);
    }
    fread(&std, sizeof(struct marks), 1, fptr);
    printf("m1: %d\tm2: %d\tm3: %d\n", std.m1, std.m2, std.m3);
    fclose(fptr);
    return 0;
}
```

Output Binary File:



Output Screen:

m1: 100 m2: 50 m3: 75

How to close a file?

The **fclose()** function is used to close an open file.

Syntax: fclose(*fp)

Ex: fclose(fp); //where fp is a file pointer

The function **fclose()** returns

- **0** on success of file close and
- **EOF** (End Of File) if any errors were detected while closing the file

---

Cursor Positioning Functions in Files

The following are the functions available in C to position cursor at a specific location in a file.

1. **ftell()**
2. **rewind()**
3. **fseek()**

ftell() - This function returns the current position of the cursor in the file.

Syntax: ftell(*file_pointer);

fseek() - This function is used to set the cursor position to the specific position.

Syntax: fseek(*file_pointer, int positions, int origin)

positions: indicates a number of characters to move the cursor to

origin: indicates starting place where the cursor position is measured

- 0 or SEEK_SET - from the beginning of the file
- 1 or SEEK_CUR- from the current cursor position
- 2 or SEEK_END- from the ending of the file

rewind() - This function is used reset the cursor position to the beginning of the file. But cannot be checked if it was successful. So, fseek() is preferred over rewind()

Syntax: rewind(*file_pointer)

Example: Program to show how to use fseek(), ftell() and rewind() functions

```
#include<stdio.h>
int main()
{
  FILE *fp;
  int position;
  fp = fopen ("file6.txt", "w+"); //w+ opens file for both Read & write
  fprintf(fp,"%s","This is my file program!");

  position = ftell(fp);
  printf("Cursor position = %d\n",position);

  rewind(fp);
  position = ftell(fp);
  printf("Cursor position = %d\n",position);

  fseek(fp,7,0);  //point at 7th position from start (0) of the file
  position = ftell(fp);
  printf("Cursor position = %d\n", position);

  fseek(fp, -3, 2);  //point at 3rd position from end (2) of the file
  position = ftell(fp);
  printf("Cursor position = %d\n", position);

  return 0;
}
```

Output:

```
Cursor position = 24
Cursor position = 0
Cursor position = 7
Cursor position = 21
```

Program: Write a C program to copy one file into another file.

```c
#include <stdio.h>
#include <stdlib.h> // To use exit()
int main()
{
FILE *fptr1, *fptr2;
char filename[100], c;
printf("Enter the filename to open for reading \n");
scanf("%s", filename);
fptr1 = fopen(filename, "r");
if (fptr1 == NULL)
{
    printf("Cannot open file %s \n", filename);
    exit(0);
}
printf("Enter the filename to open for writing \n");
scanf("%s", filename);
fptr2 = fopen(filename, "w");
if (fptr2 == NULL)
{
    printf("Cannot open file %s \n", filename);
    exit(0);
}
c = fgetc(fptr1);
while (c != EOF)
{
    fputc(c, fptr2);
    c = fgetc(fptr1);
}
printf("\nContents copied to %s", filename);
fclose(fptr1);
fclose(fptr2);
return 0;
}
```

Output:

Enter the filename to open for reading

file1.txt

Enter the filename to open for writing

file2.txt

Contents copied to file2.txt

Program: Write a C program to remove a file from the disk.

```
#include <stdio.h>
int main()
{
int status;
char fname[20];
printf("\n\n Remove a file from the disk :\n");
printf(" \n");
printf(" Input the name of file to delete : ");
scanf("%s",fname);
status=remove(fname);
if(status==0)
{
    printf("\nFile %s has been removed.",fname);
}
else
{
    printf("\nFile %s hasn't been found or does not exist.",fname);
}
return 0;
}
```

Output1:

Remove a file from the disk :
Input the name of file to delete : **file3.txt**
File file3.txt has been removed.

Output2:

Remove a file from the disk :
Input the name of file to delete : **file3.txt**
File file3.txt hasn't been found or does not exist.