In C programming, file is a place on your physical disk where information is stored.

Why files are needed?

- When a program is terminated, the entire data is lost. Storing in a file will preserve your data even if the program terminates.
- If you have to enter a large number of data, it will take a lot of time to enter them all.
  However, if you have a file containing all the data, you can easily access the contents of the file using few commands in C.
- You can easily move your data from one computer to another without any changes.

Types of Files

When dealing with files, there are two types of files you should know about:

1. Text files
2. Binary files

1. Text files

Text files are the normal .txt files that you can easily create using Notepad or any simple text editors.

When you open those files, you'll see all the contents within the file as plain text. You can easily edit or delete the contents.

They take minimum effort to maintain, are easily readable, and provide least security and takes bigger storage space.

2. Binary files

Binary files are mostly the .bin files in your computer.

Instead of storing data in plain text, they store it in the binary form (0's and 1's).

They can hold higher amount of data, are not readable easily and provides a better security than text files.

File Operations

In C, you can perform four major operations on the file, either text or binary:

1. Creating a new file
2. Opening an existing file
3. Closing a file
4. Reading from and writing information to a file

Working with files

When working with files, you need to declare a pointer of type file. This declaration is needed for communication between the file and program.

```
FILE *fptr;
```

Opening a file - for creation and edit

Opening a file is performed using the library function in the **"stdio.h"** header file: fopen().

The syntax for opening a file in standard I/O is:

```
ptr = fopen("fileopen","mode")
```

For Example:

```
fopen("E:\\cprogram\\newprogram.txt","w");



fopen("E:\\cprogram\\oldprogram.bin","rb");
```

- Let's suppose the file `newprogram.txt` doesn't exist in the location `E:\cprogram`. The first function creates a new file named `newprogram.txt` and opens it for writing as per the mode 'w'.
  The writing mode allows you to create and edit (overwrite) the contents of the file.
- Now let's suppose the second binary file `oldprogram.bin` exists in the location `E:\cprogram`. The second function opens the existing file for reading in binary mode 'rb'.
  The reading mode only allows you to read the file, you cannot write into the file.

Opening Modes in Standard I/O

File Mode	Meaning of Mode	During Inexistence of file
r	Open for reading.	If the file does not exist, fopen() returns NULL.
rb	Open for reading in binary mode.	If the file does not exist, fopen() returns NULL.
w	Open for writing.	If the file exists, its contents are overwritten. If the file does not exist, it will be created.
wb	Open for writing in binary mode.	If the file exists, its contents are overwritten. If the file does not exist, it will be created.
a	Open for append. i.e, Data is added to end of file.	If the file does not exists, it will be created.
ab	Open for append in binary mode. i.e, Data is added to end of file.	If the file does not exists, it will be created.
r+	Open for both reading and writing.	If the file does not exist, fopen() returns NULL.
rb+	Open for both reading and writing in binary mode.	If the file does not exist, fopen() returns NULL.

Opening Modes in Standard I/O		
File Mode	Meaning of Mode	During Inexistence of file
w+	Open for both reading and writing.	If the file exists, its contents are overwritten. If the file does not exist, it will be created.
wb+	Open for both reading and writing in binary mode.	If the file exists, its contents are overwritten. If the file does not exist, it will be created.
a+	Open for both reading and appending.	If the file does not exists, it will be created.
ab+	Open for both reading and appending in binary mode.	If the file does not exists, it will be created.

Closing a File

The file (both text and binary) should be closed after reading/writing.

Closing a file is performed using library function fclose().

```
fclose(fptr); //fptr is the file pointer associated with file to be closed.
```

Reading and writing to a text file

For reading and writing to a text file, we use the functions `fprintf()` and `fscanf().`

They are just the file versions of `printf()` and `scanf()`. The only difference is that, fprint and fscanf expects a pointer to the structure FILE.

Writing to a text file

Example 1: Write to a text file using fprintf()

```c
#include <stdio.h>
int main()
{
   int num;
   FILE *fptr;
   fptr = fopen("C:\\program.txt","w");

   if(fptr == NULL)
   {
      printf("Error!");
      exit(1);
   }

   printf("Enter num: ");
   scanf("%d",&num);

   fprintf(fptr,"%d",num);
   fclose(fptr);

   return 0;
}
```

This program takes a number from user and stores in the file `program.txt`.

After you compile and run this program, you can see a text file program.txt created in C drive of your computer. When you open the file, you can see the integer you entered.

Reading from a text file

Example 2: Read from a text file using fscanf()

```c
#include <stdio.h>
int main()
{
   int num;
   FILE *fptr;

   if ((fptr = fopen("C:\\program.txt","r")) == NULL){
       printf("Error! opening file");

       // Program exits if the file pointer returns NULL.
       exit(1);
   }

   fscanf(fptr,"%d", &num);

   printf("Value of n=%d", num);
   fclose(fptr);

   return 0;
}
```

This program reads the integer present in the `program.txt` file and prints it onto the screen.

If you succesfully created the file from **Example 1**, running this program will get you the integer you entered.

Other functions like `fgetchar()`, `fputc()` etc. can be used in similar way.

Reading and writing to a binary file

Functions `fread()` and `fwrite()` are used for reading from and writing to a file on the disk respectively in case of binary files.

Writing to a binary file

To write into a binary file, you need to use the function fwrite(). The functions takes four arguments: Address of data to be written in disk, Size of data to be written in disk, number of such type of data and pointer to the file where you want to write.

```
fwrite(address_data,size_data,numbers_data,pointer_to_file);
```

Example 3: Writing to a binary file using fwrite()

```
#include <stdio.h>

struct threeNum
{
   int n1, n2, n3;
};
int main()
{
   int n;
   struct threeNum num;
   FILE *fptr;

   if ((fptr = fopen("C:\\program.bin","wb")) == NULL){
       printf("Error! opening file");

       // Program exits if the file pointer returns NULL.
       exit(1);
```

```
    }
    for(n = 1; n < 5; ++n)
    {
        num.n1 = n;
        num.n2 = 5n;
        num.n3 = 5n + 1;
        fwrite(&num, sizeof(struct threeNum), 1, fptr);
    }
    fclose(fptr);

    return 0;
}
```

In this program, you create a new file `program.bin` in the C drive.

We declare a structure `threeNum` with three numbers - `n1, n2 and n3`, and define it in the main function as num.

Now, inside the for loop, we store the value into the file using fwrite.

The first parameter takes the address of `num` and the second parameter takes the size of the structure `threeNum`.

Since, we're only inserting one instance of `num`, the third parameter is `1`. And, the last parameter `*fptr` points to the file we're storing the data.

Finally, we close the file.

Reading from a binary file

Function `fread()` also take 4 arguments similar to `fwrite()` function as above.

```
fread(address_data,size_data,numbers_data,pointer_to_file);
```

Example 4: Reading from a binary file using fread()

```
#include <stdio.h>

struct threeNum
{
   int n1, n2, n3;
};

int main()
{
   int n;
   struct threeNum num;
   FILE *fptr;

   if ((fptr = fopen("C:\\program.bin","rb")) == NULL){
       printf("Error! opening file");

       // Program exits if the file pointer returns NULL.
       exit(1);
   }

   for(n = 1; n < 5; ++n)
   {
```

```
        fread(&num, sizeof(struct threeNum), 1, fptr);
        printf("n1: %d\tn2: %d\tn3: %d", num.n1, num.n2, num.n3);
    }
    fclose(fptr);

    return 0;
}
```

In this program, you read the same file `program.bin` and loop through the records one by one.

In simple terms, you read one `threeNum` record of `threeNum` size from the file pointed by `*fptr` into the structure `num`.

You'll get the same records you inserted in Example 3.

Getting data using fseek()

If you have many records inside a file and need to access a record at a specific position, you need to loop through all the records before it to get the record.

This will waste a lot of memory and operation time. An easier way to get to the required data can be achieved using fseek().

As the name suggests, fseek() seeks the cursor to the given record in the file.

Syntax of fseek()

```
fseek(FILE * stream, long int offset, int whence)
```

The first parameter stream is the pointer to the file. The second parameter is the position of the record to be found, and the third parameter specifies the location where the offset starts.

Different Whence in fseek

Whence	Meaning
SEKK_SET	Starts the offset from the beginning of the file.
SEKK_END	Starts the offset from the end of the file.
SEKK_CUR	Starts the offset from the current location of the cursor in the file.

Example of fseek()

```
#include <stdio.h>

struct threeNum
{
   int n1, n2, n3;
};

int main()
{
   int n;
   struct threeNum num;
   FILE *fptr;

   if ((fptr = fopen("C:\\program.bin","rb")) == NULL){
       printf("Error! opening file");

       // Program exits if the file pointer returns NULL.
       exit(1);
   }

   // Moves the cursor to the end of the file
```

```
   fseek(fptr, sizeof(struct threeNum), SEEK_END);

   for(n = 1; n < 5; ++n)
   {
      fread(&num, sizeof(struct threeNum), 1, fptr);
      printf("n1: %d\tn2: %d\tn3: %d", num.n1, num.n2, num.n3);
   }
   fclose(fptr);

   return 0;
}
```

This program will start reading the records from the file `program.bin` in the reverse order (last to first) and prints it