**File Handling in C**

A 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](https://www.programiz.com/c-programming/library-function) in the **"stdio.h"** header file: fopen().

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

fptr = 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. |
| 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.

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

#include <stdio.h>

#include <stdlib.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\n",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.

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

#include <stdio.h>

#include <stdlib.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 successfully 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.

**fputc**

**int fputc(int char, FILE \*pointer)**

**char:**  character to be written.

This is passed as its int promotion.

**pointer:** pointer to a FILE object that identifies the

stream where the character is to be written.

**fgetc**

**int fgetc(FILE \*pointer)**

**pointer:** pointer to a FILE object that identifies

the stream on which the operation is to be performed.

fgets()

char \*fgets(char \*str, int n, FILE \*stream)

Parameters

* **str** − This is the pointer to an array of chars where the string read is stored.
* **n** − This is the maximum number of characters to be read (including the final null-character). Usually, the length of the array passed as str is used.
* **stream** − This is the pointer to a FILE object that identifies the stream where characters are read from.

Writing File : fputs() function

The fputs() function writes a line of characters into file. It outputs string to a stream.

**Syntax:**

**int** fputs(**const** **char** \*s, **FILE** \*stream)

**Example:**

#include<stdio.h>

#include<conio.h>

**void** main(){

**FILE** \*fp;

clrscr();

fp=fopen("myfile2.txt","w");

fputs("hello c programming",fp);

fclose(fp);

getch();

}

**The fwrite() function**

The fwrite() function is used to write records (sequence of bytes) to the file. A record may be an array or a structure.

**Syntax of fwrite() function**

fwrite( ptr, int size, int n, FILE \*fp );

The fwrite() function takes four arguments.  
ptr : ptr is the reference of an array or a structure stored in memory.  
size : size is the total number of bytes to be written.  
n : n is number of times a record will be written.  
FILE\* : FILE\* is a file where the records will be written in binary mode.

**Example of fwrite() function**

#include<stdio.h>

struct Student

{

int roll;

char name[25];

float marks;

};

void main()

{

FILE \*fp;

char ch;

struct Student Stu;

fp = fopen("Student.dat","w"); //Statement 1

if(fp == NULL)

{

printf("\nCan't open file or file doesn't exist.");

exit(0);

}

do

{

printf("\nEnter Roll : ");

scanf("%d",&Stu.roll);

printf("Enter Name : ");

scanf("%s",Stu.name);

printf("Enter Marks : ");

scanf("%f",&Stu.marks);

fwrite(&Stu,sizeof(Stu),1,fp);

printf("\nDo you want to add another data (y/n) : ");

ch = getche();

}while(ch=='y' || ch=='Y');

printf("\nData written successfully...");

fclose(fp);

}

Output :

Enter Roll : 1

Enter Name : Ashish

Enter Marks : 78.53

Do you want to add another data (y/n) : y

Enter Roll : 2

Enter Name : Kaushal

Enter Marks : 72.65

Do you want to add another data (y/n) : y

Enter Roll : 3

Enter Name : Vishwas

Enter Marks : 82.65

Do you want to add another data (y/n) : n

Data written successfully...

**The fread() function**

The fread() function is used to read bytes form the file.

**Syntax of fread() function**

fread( ptr, int size, int n, FILE \*fp );

The fread() function takes four arguments.  
ptr : ptr is the reference of an array or a structure where data will be stored after reading.  
size : size is the total number of bytes to be read from file.  
n : n is number of times a record will be read.  
FILE\* : FILE\* is a file where the records will be read.

**Example of fread() function**

#include<stdio.h>

struct Student

{

int roll;

char name[25];

float marks;

};

void main()

{

FILE \*fp;

char ch;

struct Student Stu;

fp = fopen("Student.dat","r"); //Statement 1

if(fp == NULL)

{

printf("\nCan't open file or file doesn't exist.");

exit(0);

}

printf("\n\tRoll\tName\tMarks\n");

while(fread(&Stu,sizeof(Stu),1,fp)>0)

printf("\n\t%d\t%s\t%f",Stu.roll,Stu.name,Stu.marks);

fclose(fp);

}

Output :

Roll Name Marks

1 Ashish 78.53

2 Kaushal 72.65

3 Vishwas 82.65

**Random Access To File**

The C library function **int fseek(FILE \*stream, long int offset, int whence)** sets the file position of the **stream** to the given **offset**.

Declaration

Following is the declaration for fseek() function.

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

Parameters

* **stream** − This is the pointer to a FILE object that identifies the stream.
* **offset** − This is the number of bytes to offset from whence.
* **whence** − This is the position from where offset is added. It is specified by one of the following constants −

|  |  |
| --- | --- |
| **Sr.No.** | **Constant & Description** |
| 1 | **SEEK\_SET**  Beginning of file |
| 2 | **SEEK\_CUR**  Current position of the file pointer |
| 3 | **SEEK\_END**  End of file |

Example program for fseek():

**Write a program to read last ‘n’ characters of the file using appropriate file functions(Here we need fseek() and fgetc()).**

|  |  |
| --- | --- |
| 01 | #include<stdio.h> |
| 02 | #include<conio.h> |

|  |  |  |
| --- | --- | --- |
| 03 | void main() | |
| 04 | { |

|  |  |  |
| --- | --- | --- |
| 05 | FILE \*fp; | |
| 06 | char ch; |

|  |  |
| --- | --- |
| 07 | clrscr(); |
| 08 | fp=fopen("file1.c", "r"); | |

|  |  |
| --- | --- |
| 09 | if(fp==NULL) |
| 10 | printf("file cannot be opened"); | |

|  |  |  |
| --- | --- | --- |
| 11 | else | |
| 12 | { |

|  |  |  |
| --- | --- | --- |
| 13 | printf("Enter value of n  to read last ‘n’ characters"); | |
| 14 | scanf("%d",&n); |

|  |  |
| --- | --- |
| 15 | fseek(fp,-n,2); |
| 16 | while((ch=fgetc(fp))!=EOF) | |

|  |  |
| --- | --- |
| 17 | { |
| 18 | printf("%c\t",ch); | |

|  |  |  |
| --- | --- | --- |
| 19 | } | |
| 20 | } |

|  |  |  |
| --- | --- | --- |
| 21 | fclose(fp); | |
| 22 | getch(); |

|  |  |
| --- | --- |
| 23 | } |

**OUTPUT:** It depends on the content in the file.

Error Handling in C

## feof(), ferror(), clearerr()

Determine if a file has reached end-of-file or if an error has occurred.

### Prototypes

#include <stdio.h>

int feof(FILE \*stream);

int ferror(FILE \*stream);

void clearerr(FILE \*stream);

### Description

Each FILE\* that you use to read and write data from and to a file contains flags that the system sets when certain events occur. If you get an error, it sets the error flag; if you reach the end of the file during a read, it sets the EOF flag. Pretty simple really.

The functions **feof()** and **ferror()** give you a simple way to test these flags: they'll return non-zero (true) if they're set.

Once the flags are set for a particular stream, they stay that way until you call **clearerr()** to clear them.

### Return Value

**feof()** and **ferror()** return non-zero (true) if the file has reached EOF or there has been an error, respectively.

### Example

// read binary data, checking for eof or error

int main(void)

{

int a;

FILE \*fp;

fp = fopen("binaryints.dat", "rb");

// read single ints at a time, stopping on EOF or error:

while(fread(&a, sizeof(int), 1, fp), !feof(fp) && !ferror(fp)) {

printf("I read %d\n", a);

}

if (feof(fp))

printf("End of file was reached.\n");

if (ferror(fp))

printf("An error occurred.\n");

fclose(fp);

return 0;

}

Important Program of File Handling

1. Program to copy content of one file to another file(making duplicate from original)

#include<stdio.h>

#include<conio.h>

int main()

{

char ch, fileName1[20], fileName2[20];

FILE \*fs, \*ft;

printf("Enter Source File Name (with extension): ");

gets(fileName1);

fs = fopen(fileName1, "r");

if(fs == NULL)

{

printf("\nError in Opening the file, %s", fileName1);

getch();

return 0;

}

printf("Enter Target File Name (with extension): ");

gets(fileName2);

ft = fopen(fileName2, "w");

if(ft == NULL)

{

printf("\nError in Opening the file, %s", fileName2);

getch();

return 0;

}

ch = fgetc(fs);

while(ch != EOF)

{

fputc(ch, ft);

ch = fgetc(fs);

}

printf("\nFile copied successfully.");

fclose(fs);

fclose(ft);

getch();

return 0;

}

1. **Write a program to C to write integers in first file, then read contents from the same file and write the odd numbers in second file and write even integers in the third file.**

#include<stdio.h>

//#include<conio.h>

int main()

{

FILE \*fptr1,\*fptr2,\*fptr3;

int n, i, num;

printf("Enter number of values : ");

scanf("%d",&n);

printf("\nEnter the values : ");

fptr1 = fopen("NUMBERS.DAT", "w");

for(i = 0 ; i < n ; i++)

{

scanf("%d",&num);

fprintf(fptr1,"%d\n",num);

}

fclose(fptr1);

fptr1 = fopen("NUMBERS.DAT","r");

fptr2 = fopen("ODD.DAT","w");

fptr3 = fopen("EVEN.DAT","w");

while((fscanf(fptr1,"%d",&num)) != EOF)

{

if(num % 2 == 0){

putw(num, fptr3) ;

} else{

putw(num, fptr2) ;

}

}

fclose(fptr1);

fclose(fptr2);

fclose(fptr3);

fptr2 = fopen("ODD.DAT", "r");

fptr3 = fopen("EVEN.DAT", "r");

printf("\nContents of ODD file is : ");

while((num = getw(fptr2)) != EOF)

{

printf("%d\t", num);

}

printf("\n\nContents of EVEN file is : ") ;

while((num = getw(fptr3)) != EOF)

{

printf("%d\t", num);

}

fclose(fptr2);

fclose(fptr3);

}

1. **Write a program to write integers to first file, read the content of the same file and display only the even integers.**

#include <stdio.h>

int main( )

{

// creating a FILE variable

FILE \*fptr;

// integer variable

int num;

// open the file in write mode

fptr = fopen("integers", "w");

if (fptr != NULL) {

printf("File created successfully!\n");

}

else {

printf("Failed to create the file.\n");

// exit status for OS that an error occurred

return -1;

}

// enter integer numbers

printf("Enter some integer numbers [Enter -1 to exit]: ");

while (1) {

scanf("%d", &num);

if (num != -1) {

putw(num, fptr);

}

else {

break;

}

}

// close connection

fclose(fptr);

// open file for reading

fptr = fopen("integers", "r");

// display numbers

printf("\nNumbers:\n");

while ( (num = getw(fptr)) != EOF ) {

if(num%2==0)

{

printf("%d\n", num);

}

printf("\nEnd of file.\n");

// close connection

fclose(fptr);

return 0;

}