**AC File Handling**

.

A file is a container in computer storage devices used for storing data.

**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 a 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. You can easily create text files using any simple text editors such as Notepad.

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

**1. Differences between Text and Binary file**

* A **text file** stores data in the form of alphabets, digits and other special symbols by storing their ASCII values and are in a human readable format. For example, any file with a .txt, .c, etc extension. Whereas, a **binary file** contains a sequence or a collection of bytes which are not in a human readable format. For example, files with .exe, .mp3, etc extension. It represents custom data.
* A small error in a **textual file** can be recognized and eliminated when seen. Whereas, a small error in a **binary file** corrupts the file and is not easy to detect.
* Since the data is not human readable it also adds to the security of the content as one might not be able to get data if the structure is not known.
* Now, when it comes to programming there are three major differences between the two, i.e., ***Handling of newlines, storage of numbers and representation of EOF(End of File)***. Let’s look into these differences in detail:

**2. Handling of Newlines**

***Newline*** is the *end of the line* or *line ending* or*line break*. It is usually a special character which signifies the end of the line. A **newline** character in a text file is first converted into a carriage return-linefeed combination and then written to the disk. Similarly, when read by a **text file** the carriage return-linefeed combination is converted into a **newline**. However, in a **binary file**, no such conversions take place.

**3. Storage of Numbers**

In the**text mode**, the function *fprintf()* is used to store numerical data in the disk. The texts and the characters are stored one character per byte as it should be (char occupies 1 byte in the memory) and as expected the integers should occupy 4 bytes(depends on the compiler) per number. But this is not the case. For example, we have a number 567392. According to integer storage convention, it should occupy 4 bytes in the disk but it does not. It occupies 6 bytes,i.e., 1 byte for every digit in the number. Also, the number 56.9057 will occupy 7 bytes in the disk. Thus, we see that each digit in the file is treated as a character in itself and occupies more space than necessary. So, if we have a lot of numerical data then using a text file will not be very memory efficient( but still the syntax used depends on our usage i.e. if we have uses in which a human has to read the file then we can never choose the binary type).

This problem can be solved by using **binary files**. We should open the file in binary mode(using **“wb”** or **“rb”** for write and read mode respectively). The, using the function *fread()* or *fwrite()* we can easily store the data in the binary form which shall use only 4 bytes for storing the integer data.

**File Operations**

In C, you can perform four major operations on files, 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 the program.

FILE \*fptr;

**Opening a file - for creation and edit**

Opening a file is performed using the fopen() function defined in the stdio.h header file.

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

fptr = fopen("fileopen","mode");

For example,

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

bptr= 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 | | |
| --- | --- | --- |
| 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. Data is added to the end of the file. | If the file does not exist, it will be created. |
| ab | Open for append in binary mode. Data is added to the end of the file. | If the file does not exist, 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 exist, it will be created. |
| ab+ | Open for both reading and appending in binary mode. | If the file does not exist, 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 the fclose() function.

fclose(fptr);

Here, fptr is a file pointer associated with the 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**

#include <stdio.h>

#include <stdlib.h>

int main()

{

int num;

FILE \*fptr;

// use appropriate location if you are using MacOS or Linux

fptr = fopen("C:\\program.txt","w");

if(fptr == NULL)

{

printf("Error!");

exit(1);

}

printf("Enter num: ");

scanf("%d",&num);

fprintf(fptr,"%d",num); //fprintf(fptr,“%d”,num);

fclose(fptr);

return 0;

}

This program takes a number from the 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**

#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); // 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 a 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 fwrite() function. The functions take four arguments:

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.

fwrite(addressData, sizeData, numbersData, pointerToFile);

**Example 3: Write to a binary file using fwrite()**

#include <stdio.h>

#include <stdlib.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 = 5\*n;

num.n3 = 5\*n + 1;

fwrite(&num, sizeof(struct threeNum), 1, fptr);

}

fclose(fptr);

return 0;

}

In this program, we 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 the fwrite() function as above.

fread(addressData, sizeData, numbersData, pointerToFile);

**Example 4: Read from a binary file using fread()**

#include <stdio.h>

#include <stdlib.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 |
| SEEK\_SET | Starts the offset from the beginning of the file. |
| SEEK\_END | Starts the offset from the end of the file. |
| SEEK\_CUR | Starts the offset from the current location of the cursor in the file. |

**Example 5: fseek()**

#include <stdio.h>

#include <stdlib.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\n", num.n1, num.n2, num.n3);

fseek(fptr, -2\*sizeof(struct threeNum), SEEK\_CUR);

}

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.

**[Previous Tutorial:](https://www.programiz.com/c-programming/c-structure-examples" \o "C struct Examples)**