File I/O in C programming

with examples

In this guide, we will learn how to perform input/output(I/O) operations on a file using **C programming** language.

C File I/O – Table of Contents

Before we discuss each operation in detail, lets take a simple C program:

A Simple C Program to open, read and close the file

```
#include <stdio.h>
int main()
{
      /* Pointer to the file */
      FILE *fp1;
      /* Character variable to read the content of file */
      char c;

      /* Opening a file in r mode*/
      fp1= fopen ("C:\\myfiles\\newfile.txt", "r");

      /* Infinite loop -I have used break to come out of the loop*/
      while(1)
      {
         c = fgetc(fp1);
         if(c==EOF)
             break;
         else
             printf("%c", c);
      }
      fclose(fp1);
      return 0;
}
```

In the above program, we are opening a file `newfile.txt` in `r` mode, reading the content of the file and displaying it on the console. lets understand the each operation in detail:

1. Opening a file

`fopen()` function is used for opening a file.
Syntax:

```
FILE pointer_name = fopen ("file_name", "Mode");
```

`pointer_name` can be anything of your choice.
`file_name` is the name of the file, which you want to open. Specify the full path here like "C:\\myfiles\\newfile.txt".

While opening a file, you need to specify the mode. The mode that we use to read a file is "r" which is "read only mode".
for example:

```
FILE  *fp;
fp = fopen("C:\\myfiles\\newfile.txt", "r");
```

The address of the first character is stored in `pointer fp`.

How to check whether the file has opened successfully?
If file does not open successfully then the pointer will be assigned a NULL value, so you can write the logic like this:
This code will check whether the file has opened successfully or not. If the file does not open, this will display an error message to the user.

```
..
FILE fpr;
fpr = fopen("C:\\myfiles\\newfile.txt", "r");
if (fpr == NULL)
{
    puts("Error while opening file");
    exit();
}
```

Various File Opening Modes:

The file is opened using `fopen()` function, while opening you can use any of the following mode as per the requirement.
Mode "r": It is a read only mode, which means if the file is opened in r mode, it won't allow you to write and modify content of it. When `fopen()` opens a file successfully then it returns the address of first character of the file, otherwise it returns NULL.

Mode "w": It is a write only mode. The `fopen()` function creates a new file when the specified file doesn't exist and if it fails to open file then it returns NULL.

Mode "a": Using this mode Content can be appended at the end of an existing file. Like Mode "w", `fopen()` creates a new file if it file doesn't exist. On unsuccessful open it returns NULL.
File Pointer points to: last character of the file.

Mode "r+": This mode is same as mode "r"; however you can perform various operations on the file opened in this mode. You are allowed to read, write and modify the content of file opened in "r+" mode.
File Pointer points to: First character of the file.

Mode "w+": Same as mode "w" apart from operations, which can be performed; the file can be read, write and modified in this mode.
Mode "a+": Same as mode "a"; you can read and append the data in the file, however content modification is not allowed in this mode.

2. Reading a File

To read the file, we must open it first using any of the mode, for example if you only want to read the file then open it in "r" mode. Based on the mode selected during file opening, we are allowed to perform certain operations on the file.

C Program to read a file

fgetc( ): This function reads the character from current pointer's position and upon successful read moves the pointer to next character in the file. Once the pointers reaches to the end of the file, this function returns **EOF (End of File)**. We have used **EOF** in our program to determine the end of the file.

```
#include <stdio.h>
int main()
{
     /* Pointer to the file */
     FILE *fp1;
     /* Character variable to read the content of file */
     char c;

     /* Opening a file in r mode*/
     fp1= fopen ("C:\\myfiles\\newfile.txt", "r");

     /* Infinite loop -I have used break to come out of the loop*/
     while(1)
     {
        c = fgetc(fp1);
        if(c==EOF)
            break;
        else
            printf("%c", c);
     }
     fclose(fp1);
     return 0;
}
```

3. Writing to a file

To write the file, we must open the file in a mode that supports writing. For example, if you open a file in “r” mode, you won’t be able to write the file as “r” is read only mode that only allows reading.

Example: C Program to write the file

This program asks the user to enter a character and writes that character at the end of the file. If the file doesn’t exist then this program will create a file with the specified name and writes the input character into the file.

```
#include <stdio.h>
int main()
{
   char ch;
   FILE *fpw;
   fpw = fopen("C:\\newfile.txt","w");

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

   printf("Enter any character: ");
   scanf("%c",&ch);

   /* You can also use fputc(ch, fpw);*/
   fprintf(fpw,"%c",ch);
   fclose(fpw);

   return 0;
}
```

4. Closing a file

```
fclose(fp);
```

The **fclose( )** function is used for closing an opened file. As an argument you must provide a pointer to the file that you want to close.

An example to show Open, read, write and close operation in C

```
#include <stdio.h>
int main()
{
    char ch;

    /* Pointer for both the file*/
    FILE *fpr, *fpw;
    /* Opening file FILE1.C in “r” mode for reading */
    fpr = fopen("C:\\file1.txt", "r");
```

```
    /* Ensure FILE1.C opened successfully*/
    if (fpr == NULL)
    {
         puts("Input file cannot be opened");
    }

    /* Opening file FILE2.C in "w" mode for writing*/
    fpw= fopen("C:\\file2.txt", "w");

    /* Ensure FILE2.C opened successfully*/
    if (fpw == NULL)
    {
       puts("Output file cannot be opened");
    }

    /*Read & Write Logic*/
    while(1)
    {
        ch = fgetc(fpr);
        if (ch==EOF)
            break;
        else
            fputc(ch, fpw);
    }

    /* Closing both the files */
    fclose(fpr);
    fclose(fpw);

    return 0;
}
```

How to read/ write (I/O) Strings in Files – fgets & fputs

Here we will discuss how to read and write strings into a file.

```
char *fgets(char *s, int rec_len, FILE *fpr)
```

s: Array of characters to store strings.
rec_len: Length of the input record.
fpr: Pointer to the input file.
Lets take an example:

Example to read the strings from a file in C programming

```
#include <stdio.h>
int main()
{
    FILE *fpr;
    /*Char array to store string */
    char str[100];
    /*Opening the file in "r" mode*/
    fpr = fopen("C:\\mynewtextfile.txt", "r");

    /*Error handling for file open*/
```

```
    if (fpr == NULL)
    {
       puts("Issue in opening the input file");
    }

    /*Loop for reading the file till end*/
    while(1)
    {
       if(fgets(str, 10, fpr) ==NULL)
            break;
       else
            printf("%s", str);
    }
    /*Closing the input file after reading*/
    fclose(fpr);
    return 0;
}
```

In the above example we have used fgets function like this:

```
fgets(str, 10, fpr)
```

Here **str** represents the string (array of char) in which you are storing the string after reading it from file.
10 is the length of the string that needs to be read every time.
fpr is pointer to file, which is going to be read.

Why I used if(fgets(str, 10, fpr)==NULL as a logic to determine end of the file?
In the above examples, we have used ch==EOF to get to know the end of the file. Here we have used this logic because fgets returns NULL when there is no more records are available to be read.

C Program – Writing string to a file

```
int fputs ( const char * s, FILE * fpw );
```

`char *s` – Array of char.
`FILE *fpw` – Pointer (of FILE type) to the file, which is going to be written.

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

     /*Char array to store strings */
     char str[100];

     /*Opening the file FILEW.TXT in "w" mode for writing*/
     fpw = fopen("C:\\mynewtextfile2.txt", "w");

     /*Error handling for output file*/
     if (fpw== NULL)
     {
          puts("Issue in opening the Output file");
     }

     printf("Enter your string:");
```

```
    /*Stored the input string into array - str*/
    gets(str);

    /* Copied the content of str into file -
     * mynewtextfile2.txt using pointer - fpw
     */
    fputs(str, fpw);

    /*Closing the Output file after successful writing*/
    fclose(fpw);
    return 0;
}
```

fputs takes two arguments –

```
fputs(str, fpw)
```

str – str represents the array, in which string is stored.
fpw – FILE pointer to the output file, in which record needs to be written.

Point to note about fputs:
fputs by default doesn't add new line after writing each record, in order to do that manually – you can have the following statement after each write to the file.

```
fputs("\n", fpw);
```

C FILE I/O for Binary files

So far, we have learned file operations on text files, what if the files are binary (such as .exe file). The above programs will not work for binary files, however there is a minor change in handling Binary files. The main difference is the **file name & modes.** Lets understand this with the help of an example. Lets say I have two binary files bin1.exe & bin2.exe – I want to copy content of bin1.exe to bin2.exe:

Example: Reading and Writing Binary Files in C

```
#include <stdio.h>
int main()
{
    char ch;

    /* Pointers for both binary files*/
    FILE *fpbr, *fpbw;

    /* Open for bin1.exe file in rb mode */
    fpbr = fopen("bin1.exe", "rb");

    /* test logic for successful open*/
    if (fpbr == NULL)
    {
        puts("Input Binary file is having issues while opening");
    }

    /* Opening file bin2.exe in "wb" mode for writing*/
```

```
    fpbw= fopen("bin2.exe", "wb");

    /* Ensure bin2.exe opened successfully*/
    if (fpbw == NULL)
    {
       puts("Output binary file is having issues while opening");
    }

    /*Read & Write Logic for binary files*/
    while(1)
    {
        ch = fgetc(fpbr);
        if (ch==EOF)
             break;
         else
             fputc(ch, fpbw);
     }

     /* Closing both the binary files */
     fclose(fpbr);
     fclose(fpbw);

     return 0;
}
```

Note: File opening modes are “rb” and “wb” instead of “r” & “w”..

@https://beginnersbook.com/2014/01/c-file-io/