

File Handling in C

We frequently use files for storing information which can be processed by our programs. In order to store information permanently and retrieve it we need to use files.

Files are not only used for data. Our programs are also stored in files.

The editor which you use to enter your program and save it, simply manipulates files for you.

The Unix commands `cat`, `cp`, `cmp` are all programs which process your files.

In order to use files we have to learn about **File I/O** i.e. how to write information to a file and how to read information from a file.

We will see that file I/O is almost identical to the terminal I/O that we have been using so far.

The primary difference between manipulating files and doing terminal I/O is that we must specify in our programs which files we wish to use.

As you know, you can have many files on your disk. If you wish to use a file in your programs, then you must specify which file or files you wish to use.

Specifying the file you wish to use is referred to as **opening** the file.

When you open a file you must also specify what you wish to do with it i.e. **Read** from the file, **Write** to the file, or both.

Because you may use a number of different files in your program, you must specify when reading or writing which file you wish to use. This is accomplished by using a variable called a **file pointer**.

Every file you open has its own file pointer variable. When you wish to write to a file you specify the file by using its file pointer variable.

You declare these file pointer variables as follows:

```
FILE *fopen(), *fp1, *fp2, *fp3;
```

The variables `fp1`, `fp2`, `fp3` are file pointers. You may use any name you wish.

The file `<stdio.h>` contains declarations for the Standard I/O library and should always be **included** at the very beginning of C programs using files.

Constants such as `FILE`, `EOF` and `NULL` are defined in `<stdio.h>`.

You should note that a file pointer is simply a variable like an integer or character.

It does **not** *point* to a file or the data in a file. It is simply used to indicate which file your I/O operation refers to.

A file number is used in the Basic language and a unit number is used in Fortran for the same purpose.

The function **fopen** is one of the Standard Library functions and returns a file pointer which you use to refer to the file you have opened e.g.

```
fp = fopen( "prog.c", "r" ) ; /* if r is not supporting use
read* /
```

The above statement **opens** a file called prog.c for **reading** and associates the file pointer fp with the file.

When we wish to access this file for I/O, we use the file pointer variable fp to refer to it.

You can have up to about 20 files open in your program - you need one file pointer for each file you intend to use.

File I/O

The Standard I/O Library provides similar routines for file I/O to those used for standard I/O.

The routine `getc(fp)` is similar to `getchar()`
and `putc(c,fp)` is similar to `putchar(c)`.

Thus the statement

```
c = getc(fp);
```

reads the next character from the file referenced by fp and the statement

```
putc(c,fp);
```

writes the character c into file referenced by fp.

```
/* file.c: Display contents of a file on screen */
```

```
#include <stdio.h>
```

```
void main()
```

```
{
```

```
    FILE *fopen(), *fp;
```

```
    int c ;
```

```
    fp = fopen( "prog.c", "r" );
```

```
    c = getc( fp ) ;
```

```
    while ( c != EOF )
```

```
    {
```

```
        putchar( c );
```

```

        c = getc ( fp );
    }

    fclose( fp );
}

```

In this program, we open the file `prog.c` for reading.

We then read a character from the file. This file must exist for this program to work.

If the file is empty, we are at the end, so `getc` returns EOF a special value to indicate that the end of file has been reached. (Normally -1 is used for EOF)

The while loop simply keeps reading characters from the file and displaying them, until the end of the file is reached.

The function **`fclose`** is used to *close* the file i.e. indicate that we are finished processing this file. We could reuse the file pointer `fp` by opening another file.

This program is in effect a special purpose `cat` command. It displays file contents on the screen, but **only** for a file called `prog.c`.

By allowing the user enter a file name, which would be stored in a string, we can modify the above to make it an **interactive** `cat` command:

```

/* cat2.c: Prompt user for filename and display file on screen */

#include <stdio.h>

void main()
{
    FILE *fopen(), *fp;
    int c ;
    char filename[40] ;

    printf("Enter file to be displayed: ");
    gets( filename ) ;

    fp = fopen( filename, "r");/* works with both r and read in
C++ */

    c = getc( fp ) ;

    while ( c != EOF )
    {
        putchar(c);
        c = getc ( fp );
    }

    fclose( fp );
}

```

```
}
```

In this program, we pass the name of the file to be opened which is stored in the array called `filename`, to the `fopen` function. In general, anywhere a string constant such as `"prog.c"` can be used so can a character array such as `filename`. (Note the **reverse** is **not** true).

The above programs suffer a major limitation. They **do not** check whether the files to be used exist or not.

If you attempt to read from a non-existent file, your program will crash!!

The `fopen` function was designed to cope with this eventuality. It checks if the file can be opened appropriately. If the file **cannot be opened**, it returns a **NULL** pointer. Thus by checking the file pointer returned by `fopen`, you can determine if the file was opened correctly and take appropriate action e.g.

```
fp = fopen (filename, "r") ;

if ( fp == NULL)
{
    printf("Cannot open %s for reading \n", filename );
    exit(1) ; /*Terminate program: Commit suicide !!*/
}
```

The above code fragment shows how a program might check if a file could be opened appropriately.

The function `exit()` is a special function which terminates your program immediately.

`exit(0)` means that you wish to indicate that your program terminated successfully whereas a nonzero value means that your program is terminating due to an error condition.

Alternatively, you could prompt the user to enter the filename again, and try to open it again:

```
fp = fopen (fname, "r") ;

while ( fp == NULL)
{
    printf("Cannot open %s for reading \n", fname );

    printf("\n\nEnter filename : " );
    gets( fname );

    fp = fopen (fname, "r") ;
}
```

In this code fragment, we keep reading filenames from the user until a valid existing filename is entered.

Exercise: Modify the above code fragment to allow the user 3 chances to enter a valid filename. If a valid file name is not entered after 3 chances, terminate the program.

RULE: Always check when opening files, that fopen succeeds in opening the files appropriately.

Obeying this simple rule will save you much heartache.

Program 1: Write a program to count the number of lines and characters in a file.

Note: Each line of input from a file or keyboard will be terminated by the newline character '\n'. Thus by counting newlines we know how many lines there are in our input.

Program 2: Write a program to display file contents 20 lines at a time. The program pauses after displaying 20 lines until the user presses either Q to quit or Return to display the next 20 lines. (The Unix operating system has a command called **more** to do this) As in previous programs, we read the filename from user and open it appropriately. We then process the file:

```
read character from file

while not end of file and not finished do
begin

    display character

    if character is newline then
        linecount = linecount + 1;

    if linecount == 20 then
begin
        linecount = 1 ;
        Prompt user and get reply;
end
    read next character from file
end
```

Program 3: Write a program to compare two files specified by the user, displaying a message indicating whether the files are identical or different. This is the basis of a **compare** command provided by most operating systems. Here our file processing loop is as follows:

```
read character ca from file A;
read character cb from file B;

while ca == cb and not EOF file A and not EOF file B
begin
    read character ca from file A;
    read character cb from file B;
end

if ca == cb then
    printout("Files identical");
```

```
else
    printout( "Files differ" );
```

Writing to Files

The previous programs have opened files for reading and read characters from them.

To write to a file, the file must be opened for writing e.g.

```
fp = fopen( fname, "w" );
```

If the file does not exist already, it will be created. If the file does exist, it will be overwritten! So, be careful when opening files for writing, in case you destroy a file unintentionally. Opening files for writing can also fail. If you try to create a file in another users directory where you do not have access you will not be allowed and fopen will fail.

Character Output to Files

The function `putc(c, fp)` writes a character to the file associated with the file pointer `fp`.

Example:

Write a file copy program which copies the file “prog.c” to “prog.old”

Outline solution:

```
Open files appropriately
Check open succeeded
Read characters from prog.c and
Write characters to prog.old until all characters      copied

Close files
```

The step: “Read characters and write ..” may be refined to:

```
read character from prog.c
while not end of file do
begin
    write character to prog.old
    read next character from prog.c
end
```