$Codecall \ \rightarrow \ Tutorial \ Forum \, s \ \rightarrow \ C/C +\!\!\!\! + Tutorials$



Reading And Writing Files In C

w rite,

Started by Guest, Nov 03 2009 07:57 PM

hello world,

read,

file,

Page 1 of 4

Guest

Posted 03 November 2009 - 07:57 PM

This tutorial will show you how to read and write files in C. All file functions need **<stdio.h>** to work properly.

The first thing you need to know about is file pointers. File pointers are like <u>any other pointer</u>
(http://forum.codecall.net/topic/52150-pointers-what-how-and-why/) but they point to a file (Kind of obvious) You

<u>and-why/)</u>, but they point to a file. (Kind of obvious). You define a file pointer as follows:

FILE *filepointer;

In order to make the file pointer point to a file you use the **fopen** function. The function works as follows:

filepointer=fopen("filename", "mode");

<u>fopen (http://forum.codecall.net/tags/forums/fopen/)</u> returns a file pointer. It returns NULL if the file does not exist. fopen takes the first argument as the filename to open. It needs to be a string.

The second argument is the mode argument Mode specifies what you want to do with the file. Some modes are:

- "r" read the file
- "w" write the file
- "a" append to the file
- "r+" read and write to the file
- "w+" read and write, overwrite the file
- "a+" read and write, append

These modes will open files in text mode. Files opened in text mode have some bytes filtered out. If you want to open binary files use binary mode by adding a "b" to the mode. For example:

• "rb" - read the file in binary mode

Dynamic Arrays: Using malloc() and realloc()
(http://forum.codecall.net/topic/51010-dynamic-arrays-using-malloc-and-realloc/)

There are three input and output streams that are automatically open whenever your program starts. These are

Recent Topics



For all our would-be entrepeneurs WingedPanther - Today, 05:06 PM



Hello Everyone:D Dr WallFlower - Today, 03:08 PM



Strong Mathematics background Useful? DrWallFlower - Today, 12:36 PM $\,$



WordPress Theme Free lora - Today, 10:02 AM



Powerful Servers in Mexico from DigitalServer| 99.9% Network Uptime, Good Pricing!

scriptchip - Today, 12:46 AM

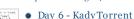
Recent Blog Entries



• Day 7 - Skype, Mutt and Nano LKP's Blog 15 Jan



• When 1,2 and 3 becomes 4, 4 and 4 in javascript lespauled's Blog 14 Jan



LKP's Blog 13 Jan



Day 5 - Electricity is shocking!
 LKP's Blog 10 Jan



• Day 4 - Media Server LKP's Blog o5 Jan

Recent Status Updates

stdin, stdout and stderr.

These file pointers work as follows:

- stdin: opened in read ("r") mode, this file pointer reads from **standard input**. Any input that you provide on the command line is read by stdin.
- stdout: opened in write ("w") mode, this file pointer writes to the standard output. Anything written to this stream is printed on the command line.
- stderr: opened in write ("w") mode, this file pointer writes to the standard error. This is generally meant for error messages that are produced by your program. Whatever is written to this stream is usually printed on the command line like stdout. In most operating systems, there are ways to tell whether output was sent to stdout or stderr.

To read a character from a file, you use **fgetc**. It is like getchar, but for any file, not just stdin.

It works like this:

character=fgetc(filepointer);

fgetc returns the character that is read from the file as an

fgetc takes the file pointer as its only input.

It will automatically increment the pointer to read the next character.

fputc allows you to write a character to a file:

fputc(character, filepointer);

fputc takes an unsigned char as the first argument and the file pointer as the second argument.

fputc returns EOF when it reaches the end of file. EOF is a constant defined in <stdio.h>

Difference between c and c++

(http://forum.codecall.net/topic/40379-difference-betweenc-and-c/)

You can also use **fprintf** and **fscanf**. They work like printf and scanf, except the file pointer is the first argument. They work like this:

fprintf(filepointer, "Hello, World!\n"); //write hello world to

fscanf(filepointer, "%d", integer); //read an integer from the f ile

In order to close the file again, you must use fclose. It looks like this:



Kadence

og Feb

Changed my name from LKP to Kadence for no real reason:D

Show comments (3)



KJGino

o8 Feb

20 Jan

"Java is related to JavaScript in the same way that a grape is related to a grapefruit."

Show comments (2)



Geek munchkin

Donovan

04 Jan Back on CodeCall, looks like I will be coding some ** sometime!

Show comments (1)



camD357

oi Jan

Happy New Year 2014 to each and every member of CodeCall, this is really a community and I am proud to belong here:D

Show comments (1)

View All Updates

Popular Tags

element syntax help html5 array generator mysql string c++ java php html timer loop C# ajax jquery form vb.net assembly

```
fclose(filepointer);
```

fclose closes the file that filepointer points to.

For example, if you want to print the contents of data.txt the code could look something like this:

```
int main()
{
FILE *filepointer;
int character;
filepointer=fopen("data.txt", "r"); /* filepointer points to dat
a.txt */
if (filepointer==NULL) { /* error opening file returns NULL
*/
printf("Could not open data.txt!\n"); /* error message */
return 1; /* exit with failure */
}
/* while character is not end of file */
while ((character=fgetc(filepointer)) != EOF) {
putchar(character); /* print the character */
}
fclose(filepointer); /* close the file */
return 0; /* success */
}
```

Getter and Setter Methods

(http://forum.codecall.net/topic/50480-getter-and-settermethods/)

There are also **fputs** and **fgets**. **fputs** is simple, similar to **puts**. Unlike **puts**, it does not automatically append a newline to supplied string. It writes a line to a file like so:

```
fputs("string\n", filepointer);
```

fgets is a special function in C. It is regarded as the best function in standard C to reliably accept input. Functions like scanf have undefined behavior when given erroneous input. Programs that use scanf or gets can have buffer overflows and be susceptible to exploits! scanf can be nice for those just starting to learn C, but it should never be used in real-world code.

fgets usage looks like this:

```
char input[100];
fgets(input, sizeof(input), filepointer);
```

The middle argument is the beauty of fgets. It is the limit of characters that fgets will store in the char array. Usually sizeof(firstargument) is a good idea, unless you are allocating memory with malloc or a similar function.

fgets reads as much as it can. If the input goes over the limit, it will stop. The next file-reading function will continue where

fgets left off. (Many times, fgets again, in a loop)

fgets does not discriminate. It reads spaces and newlines with the rest of the input.

fgets returns **NULL** when nothing can be read. (end of file) Here's an example very similar to the one above, but instead of fgetc, it uses fgets and fputs:

```
int main()
{
FILE *filepointer;
char string[100];
filepointer=fopen("data.txt", "r"); /* filepointer points to dat
a.txt */
if (filepointer==NULL) { /* error opening file returns NULL
*/
printf("Could not open data.txt!\n"); /* error message */
return 1; /* exit with failure */
}
/* while we're not at end of file */
while (fgets(string, sizeof(string), filepointer) != NULL) {
fputs(string); /* print the string */
}
fclose(filepointer); /* close the file */
return 0; /* success */
}
```

That's it! If you have questions, comments or suggestions feel free to post! +rep is very appreciated.

Looking for more beginner tutorials in C?

A simple TCP server using Linux CAPI

(http://forum.codecall.net/topic/63924-a-simple-tcp-server-

using-linux-c-api/)

Basic C, for beginners!

(http://forum.codecall.net/topic/45101-basic-c-for-

beginners/)

Switch and cases in C (http://forum.codecall.net/topic/45079-switch-and-cases-in-c/)

Edited by Roger, 26 February 2013 - 03:57 PM.

John

Posted 03 November 2009 - 08:55 PM

Nice tutorial. One thing I never fully took the time to understand is the other modes. Technically, aren't there 12 modes? r, w, a, r+, w+, a+ and then all those in binary mode? I never understood the difference between x, x+, and xb- could you elaborate on those?

```
Guest_Jordan_*
```

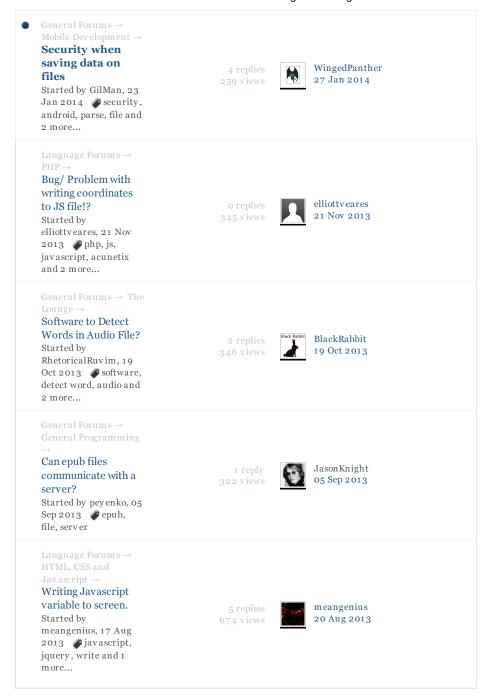
Posted 04 November 2009 - 05:12 AM

Nicely done, I've already referred a new member to this tutorial. (1)

+rep

WingedPanther	Posted 04 November 2009 - 08:40 AM
	short and sweet. +rep
so1i	Posted 04 November 2009 - 10:48 AM
	This is good, well structured. +rep
Guest	Posted 04 November 2009 - 06:19 PM
	John said
	Nice tutorial. One thing I never fully took the time to understand is the other modes. Technically, aren't there 12
	modes? r, w, a, r+, w+, a+ and then all those in binary mode? I never understood the difference between x, x+, and xb - could y ou elaborate on those?
	I have added the x+ file modes to the list, but I am not really
	sure what binary mode does. I will look into it. Edit: I have researched it and added binary mode to the tutorial.
	Edited by Guest, 04 November 2009 - 07:47 PM.
marwex89	Posted 08 November 2009 - 10:20 AM
	+rep for the tut, you beggar 🤤
debtboy	Posted 08 November 2009 -12:08 PM
	Great tutorial +rep
	Edited by debtboy, 10 November 2009 - 02:14 PM.
Guest	Posted 08 November 2009 - 02:38 PM
	Thanks everyone 🤐
tonymorrison39	Posted 21 November 2009 - 08:46 AM
	very nice i learned something
arkanion	Posted 13 January 2010 -12:00 PM
	nice issue
Phineas	Posted 13 January 2010 - 03:27 PM
	Nice TuT.

Page 1 of 4 Back to C/C++ Tutorials



 $Codecall \ \rightarrow \ Tutorial \ For um \, s \ \rightarrow \ C/C++ \ Tutorial \, s$