Read - Write File in C

The previous chapter explained the standard input and output devices handled by the C language. In this chapter we will see how programmers create, open and close text files or binary files with data. storage.

The previous chapter explained the standard input and output devices handled by the C language. In this chapter we will see how programmers create, open and close text files or binary files with data. storage.

A file represents a sequence of bytes, regardless of whether it is a text file or a binary file. The C programming language provides high-level and low-level access functions (operating system level) to manipulate files on storage devices. This chapter will take you to important function calls for file management.

Open the file in C

You can use the **fopen ()** function to create a new file or to open existing files. This call will initialize the **FILE** type object, which includes the information needed to control the flow. Here is a way to call the function:

```
FILE  * fopen ( const char *  ten_file , const char *  che_do  );
```

Here, ten_file is a string, treated as the file name and the value che_do access can be the following values:

Mode Description Open existing files for the purpose of reading w Open files for recording purposes. If these files do not yet exist, the new file file is created. Here, the program starts to write content from the beginning of file a Open the text file for writing in the next write mode at the end, if it does not already exist, the new file is created. This is a program that writes content with the end of an existing file. r + Open the text file for the purpose of reading and writing. w + Open a text file for reading and writing modes. It whitens an existing file if the file is available and creates a new one if it doesn't already exist. a + Open existing file for reading and writing purposes. It creates a new file if it doesn't exist. Reading the file will start reading from the beginning but writing the file will only write at the end of the file.
If you manipulate binary files, you can have access methods instead of the following:

```
"rb" , "wb" , "ab" , "rb+" , "r+b" , "wb+" , "w+b" , "ab+" , "a+b"
```

Close the file in C

To close a file you can use the **fclose ()** function below:

```
int  fclose (  FILE  * fp  );
```

The function **fclose ()** returns zero if success or **EOF** if there is an error in the file closing process. This function actually deletes the data in the buffer for the file, closes the file and frees the memory used with the file. **EOF** is

a constant defined in the **stdio.h** section.

There are many different functions provided by the standard library of C language to read and write each character and in a form with a fixed number of characters. We will consider in the following example:

Write to a file in C

Here is the simplest function to perform writing individual characters to a thread:

```
int  fputc ( int  c ,  FILE  * fp  );
```

The function **fputc ()** writes characters with parameter value c to an output stream referenced by the fp pointer. It returns the character written if success or **EOF** if there is an error. You can use the following function to write a string ending with null characters to a thread:

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

Function **fputs ()** writes the string **s** to an output stream referenced by fp. It returns a non-negative value if successful and returns the EOF character if an error occurs. You can use **the int fprintf function (FILE * fp, const char * format, .)** to write a string to a file. Try the example below:

You must make sure you have the / tmp directory, if not, you must create this directory on your computer.

```
#include  main () {  FILE  * fp ;  fp  =  fopen ( "vidu.txt" , "w+" );  fprintf
```

When the above code is compiled and executed, it creates a new file **vidu.txt** and writes it 2 lines of 2 different functions. Read this file in the next section.

Read file in C

Here is the simplest function to read a single character from the file:

```
int  fgetc (  FILE  *  fp  );
```

The **fgetc ()** function reads a character from a file referenced by the subclass fp. The return value is a readable character if successful, and in the case of an **EOF** return error. The following function allows you to read strings from a thread:

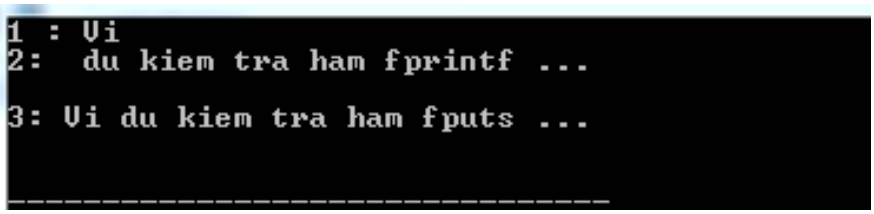
```
char * fgets ( char * buf , int  n ,  FILE  * fp  );
```

The function **fgets ()** reads n-1 characters from a thread into reference by fp. It copies the read string to the **buf** buffer, assigns the **null** character to the end of the string.

If the function encounters a newline character (new line) (nline) 'n' or EOF characters before reading the maximum number of characters, it will return only the characters until the carriage return and New line character. You can use the **int fscanf** function **(FILE * fp, const char * format, .)** to read strings from a file, but stopping reading in the first space encountered:

```
#include  main () {  FILE  * fp ; char  buff [ 255 ];  fp  =  fopen ( "vidu.txt"
```

Compile and run the above C program, it first reads from the file created from the previous area and prints the following result:



Let's see a little more detail about what happened here. First fscanf () only reads This because then it has a space, followed by the function fgets () that returns the remaining lines until the end of the file. Finally, it calls the function fgets () to read the second line completely.

Import - Export binary function

Here are two functions, which can be used for binary input and output:

```
size_t  fread ( void * ptr , size_t  kich_co_cua_cac_phan_tu , size_t  so_phan_
```

Both of these functions are used to read and write memory blocks, usually arrays or structures.

According to Tutorialspoint

Previous lesson: Input & Output in C

Next article: The preprocessor in C