C Tutorial – File I/O (using text files)

The file I/O functions and types in the C language are straightforward and easy to understand. To make use of these functions and types you have to include the stdio library. (Like we already did in most of the tutorials).

The file I/O functions in the stdio library are:

fopen – opens a text file.	**لفتح ملف**
fclose – closes a text file.	لاغلاق ملف
feof – detects end-of-file marker in a file.	اكتشاف نهاية الملف
fscanf – reads formatted input from a file.	قراءة من الملف بشكل محدد
fprintf – prints formatted output to a file.	كتابة الى الملف بشكل محدد
fgets – reads a string from a file.	قراءة سلسلة حرفية من الملف
fputs – prints a string to a file.	كتابة سلسلة حرفية الى الملف
fgetc – reads a character from a file.	قراءة حرف من الملف
fputc – prints a character to a file.	كتابة حرف الى الملف

File I/O: opening a text file فتح ملف نصي في لغة السي

The fopen library function is used to open a text file. You also have to use a specified mode when you open a file. The three most common modes used are read (r), write (w), and append (a). Take a look at an example:

```
#include<stdio.h>

int main()
{
        FILE *ptr_file;         // انشاء مؤشر للملف

        ptr_file =fopen("output.txt", "w");  //فتح الملف للكتابة

        if (!ptr_file)          //التأكد من ان الملف تم فتحه
                return 1;       //

        /*كتابة الأوامر للتعامل مع الملف في هذا الجزء */

        fclose(ptr_file);       //إغلاق الملف

        return  0;
}
```

So let's take a look at the example:

ptr_file =fopen("output", "w");

The fopen statement opens a file "output.txt" in the write (w) mode. If the file does not exist it will be created. But you must be careful! If the file exists, it will be destroyed and a new file is created instead. The fopen command returns a pointer to the file, which is stored in the variable ptr_file. If the file cannot be opened (for some reason) the variable ptr_file will contain NULL.

if (!ptr_file)

The if statement after de fopen, will check if the fopen was successful. If the fopen was not successful, the program will return a one. (Indicating that something has gone wrong).

fclose(ptr_file);

The fclose statement will close the file. This command must be given, especially when you are writing files. So don't forget it. You have to be careful that you don't type "close" instead of "fclose", because the close function exists. But the close function does not close the files correctly. (If there are a lot of files open but not closed properly, the program will eventually run out of file handles and/or memory space and crash.)

File I/O: reading a text file

If you want to read a file you have to open it for reading in the read (r) mode. Then the fgets library functions can be used to read the contents of the file. (It is also possible to make use of the library function fscanf. But you have to be sure that the file is perfectly formatted or fscanf will not handle it correctly). Let's take a look at an example:

```
#include<stdio.h>

int main()
{
        FILE *ptr_file;         // انشاء مؤشر للملف
        char buf[1000];         // سلسلة حرفية بطول الف حرف

        ptr_file =fopen("input.txt","r"); // فتح الملف للقراءة
        if (!ptr_file)          //التأكد من فتح الملف
                return 1;

        while (fgets(buf,1000, ptr_file)!=NULL) //قراءة سلسلة بطول
                                //الف حرف الى ان ينتهي الملف
                printf("%s",buf);       //طباعة السلسلة

        fclose(ptr_file);       // إغلاق الملف
        return 0;
}
```

Note: The printf statement does not have the new-line (\n) in the format string. This is not necessary because the library function fgets adds the \n to the end of each line it reads.

A file "input.txt" is opened for reading using the function fopen en the mode read (r). The library function fgets will read each line (with a maximum of 1000 characters per line.) If the end-of-file (EOF) is reached the fgets function will return a NULL value. Each line will be printed on stdout (normally your screen) until the EOF is reached. The file is then closed and the program will end.

مثال: هذا البرنامج يقوم بقراءة ملف نصي ويقوم بطباعة كل كلمة token في سطر

```
#include <stdio.h>
#include <stdlib.h>
#include <ctype.h>
#include <fcntl.h>
#include <errno.h>
#include <unistd.h>
#include <string.h>


int main ()
{
    int c;
    FILE *input_file;

    input_file = fopen("input.txt","r");

    if (input_file == 0)
    {
        //fopen returns 0, the NULL pointer, on failure
        perror("Canot open input file\n");
        exit(-1);
    }
    else
    {
        int found_word = 0;

        while ((c =fgetc(input_file)) != EOF )  //  قراءة الملف حرفا حرفا
                                                // نصل الى نهاية الملف
        {
            //if it's an alpha, convert it to lower case
            //  يتم اختبار الحرف اذا كان حرفا ابجديا
            if (isalpha(c))
            {
                found_word = 1;
                c = tolower(c);   // تغيير الحرف الى حرف صغير
                putchar(c);       // طباع الحرف
            }
            else {
                if (found_word) {    // اذا لم يكن حرفا ابجديا
                    putchar('\n');     // طباعة سطر جديد
                    found_word=0;
                }
            }

        }
    }

    fclose(input_file);

    printf("\n");

    return 0;
}
```

C Tutorial – Command Line Parameter Parsing

In this tutorial we take another look at command line parameter parsing with C programs. In a **previous command line parameter tutorial** we already looked at some simple command line argument parsing example. Please read that tutorial before advancing below.

More Complex Command Line Parameter Parsing Example

Command-line parameters can be used for many things. Let's take a look at another example that will print different things depending on the flags given. If the –f flag is used then the given argument is printed once.

If the –d flag is used the argument is printed twice.

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

void usage(void)
{
        printf("Usage:\n");
        printf(" -f<name>\n");
        printf(" -d<name>\n");
        exit (8);
}
int main(int argc, char *argv[])
{
        printf("Program name: %s\n", argv[0]);

        while ((argc > 1) && (argv[1][0] == '-'))
        {
                switch (argv[1][1])
                {
                        case 'f':
                                printf("%s\n",&argv[1][2]);
                                break;

                        case 'd':
                                printf("%s\n",&argv[1][2]);
                                printf("%s\n",&argv[1][2]);
                                break;

                        default:
                                printf("Wrong Argument: %s\n", argv[1]);
                                usage();
                }

                ++argv;
                --argc;
        }
        return (0);
}
```

After compiling you can run the program with the following parameters:

- # program
- # program –fhello
- # program –dbye
- # program –fhello –dbye
- # program –w

Note: that there is no space between the flag and the flag argument.

Closer Look at the Program

First the program name is printed (it will always be printed.)

The "while loop" looks for the dash sign. In case of the –f the flag argument is printed once. If the flag is –d then the flag argument is printed twice.

The argv[][] array will be used as follows:

For example: –f<name>

- argv[1][0] contains the dash sign
- argv[1][1] contains the "f"
- argv[1][2] contains the flag argument name

If a wrong flag sign is given then the usage is printed onto the screen.

Experiment Yourself

If you want to experiment further with command line parsing then you should expand the program above with the ability to add a space between the flag and the flag argument. Another thing you could add is parsing the program name to the usage function and prints it from this function.

مثال: هذا البرنامج يقوم بقراءة ملف نصي ويقوم بطباعة كل كلمة token في سطر – يتم قراءة الملف من سطر الاوامر

```
#include <stdio.h>
#include <stdlib.h>
#include <ctype.h>
#include <fcntl.h>
#include <errno.h>
#include <unistd.h>
#include <string.h>

int main (int argc, char **argv)
{
    int c;
    char *input = argv[1];
    FILE *input_file;

    input_file = fopen(input, "r");
```

```
    if (input_file == 0)
    {
        //fopen returns 0, the NULL pointer, on failure
        perror("Canot open input file\n");
        exit(-1);
    }
    else
    {
        int found_word = 0;

        while ((c =fgetc(input_file)) != EOF )  //  قراءة الملف حرفا حرفا
                                                // نصل الى نهاية الملف
        {
            //if it's an alpha, convert it to lower case
            //  يتم اختبار الحرف اذا كان حرفا ابجديا
            if (isalpha(c))
            {
                found_word = 1;
                c = tolower(c);   // تغيير الحرف الى حرف صغير
                putchar(c);       // طباع الحرف
            }
            else {
                if (found_word) {    // اذا لم يكن حرفا ابجديا
                    putchar('\n');     // طباعة سطر جديد
                    found_word=0;
                }
            }

        }
    }

    fclose(input_file);

    printf("\n");

    return 0;
}
```

المراجع:

تم تجميع هذا الدرس من

https://www.codingunit.com/c-tutorial-file-io-using-text-files

https://stackoverflow.com/questions/18109458/read-from-a-text-file-and-parse-lines-into-words-in-c