

Lab 3: Reading Data

CPSC 2310 – Spring 22

**Type: Programming**

**Due: Monday, February 7, 2022**

# Lab Objective

Practice with:

* reading and processing data
* fscanf, fgetc, and printf
* assert
* command line arguments
* multiple files
* header guards

**Collaboration Instructions**

* Students may work in a pair on this assignment
* A pair can be two students from the same lab section you may not go beyond your section
* Feel free to work alone
* Each student in a pair will submit to handin.
* In your comment header (where you would put your name, section, etc.) Include both students in a pairs names.
* Keep multi-pair collaboration to a minimum. Ask a TA for help for any serious answers about the lab.

# Introduction

This is a “C” program. You are not allowed to use any other programing language.

You are going to write two versions of the same program. Your program is going to read information from a file. The input file will contain a “C” program that has several valid comments.

/\*Example comment\*/ or //Example comment

Your program should detect and ignore all comments from the input file. Your program should print all remaining characters in the program to stdout (this means to the terminal).

As an example, if the input file contains the following:

#include <stdio.h>

int main()

{

/\*This is a comment\*/

int a = 0;

int b = 5;

//This is a comment

for(a = 0; a < 5; a++)

{

/\*This is another comment\*/

printf("%d\n", b);

b++;

}

//another comment

return 0;

}

Your program should print the the following to the terminal:

#include <stdio.h>

int main()

{

int a = 0;

int b = 5;

for(a = 0; a < 5; a++)

{

printf("%d\n", b);

b++;

}

return 0;

}

The input file name will be defined using command line arguments. You are required to check that the appropriate number of command line arguments were provided during execution. You may use assert or write a function to check the command line arguments. If you choose to write a function and the correct number of arguments are not provided on the command line, your function should print a statement to stdout (the terminal) indicating not enough command line arguments were provided and exit the program.

Since you are reading from a file you are required to open the file. After opening the file you are required to check that the file opened successfully. Again, you may use assert to do this or you may write a function. If you write a function, upon an unsuccessful attempt to open the file you should print a message indicating the file did not open successfully and exit the program.

You will be provided a separate set of files for each of the two programs.

Set 1 will use fgetc to read the data and Set 2 will use fscanf to read the data. These sets of programs are mostly identical programs. The only difference is one uses fscanf and the other uses fgetc. Focus on getting one set working, then change the functions that reads in the data for the other. Don’t make this hard.

Set 1: Lab3FgetcDriver.c, Lab3Fgetc.h, Lab3Fgetc.c.

Set 2: Lab3FscanfDriver.c, Lab3Fscanf.h, Lab3Fscanf.c

Drivers:

The driver should contain main. The driver should have minimal code. It should check the command line arguments, create the file pointer, check that the file pointer opened correctly, call the function start, then clean up.

Header files:

The .h file provides the function prototypes. You must use header guards in the .h files.

Implementation files:

You will implement the start, rcomment, skipM\_comment, and skipS\_comment functions in a separtate .c file.

This lab is not designed to be excessively difficult. It is designed to give you practice writing simple algorithms and using basic C programming concepts.

Other requirements:

* Your programs must compile on one of the “cerf” SoC servers.
* You must add a header to each of your files similar to the following:

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*Your names

\*CPSC 2311 Sp22 Section 00?

\*Your email

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

* Your code should be well documented. (comments)
* There should be no lines of code longer than 80 characters. This will be strictly monitored.
* You should use proper and consistent indention.

Violating any of the above will cost a minimal 5 points each. A higher number of offenses will increase the amount of point reduction.

Here are some guidelines for documenting code.

Before each function you should have a detailed description of what the overall function does. To borrow from another student’s code, here is an example of overall function description. Your description should be more than just a word or two.

/\* Parameters: img - image\_t pointer array holding the image data for  
 \*                   each of the input files  
 \* Return:     output - image\_t struct containing output image data  
 \* This function averages every pixels rbg values from each of the   
 \* input images and puts those averages into a single output image …  
 \*/

Also, if you include comments in the body of the function (and you should) they should be placed above the line of code not beside the code.

Example:

Good

//This is a comment

if(something)

{

do something;

}

Bad

if(something) //This is a comment

{

do something;

}

Handin:

You will use the School of Computing handin system to submit your program files.

You are required to tar zip your files calling the tarred file: <username>\_lab3.tar.gz, where <username> is replaced with your Clemson username. As an example, my tarred file would be called: yfeaste\_lab3.tar.gz. Be sure to check your submission by downloading your tarred file after submitting it to handin, untar the file, and examine the files to determine there were no files corrupt during the tarring process.