**Learning Objectives:**

This assignment is designed to provide practice with the following:

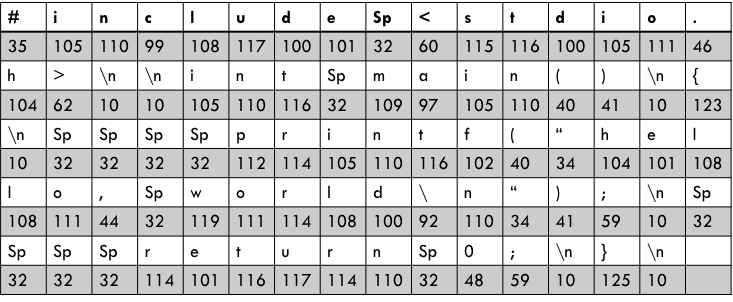
* **fread**
* **file I/O**
* **fseek**
* **file I/O macros**
* **general C programming concepts**

**Overview:**

**Read the entire document to ensure you are aware of all requirements. I will not accept the excuse you did not know you were supposed to do something.**

This is an individual assignment, you are not allowed to receive help from anyone other than, myself, or the 2310 lab TA’s. There are several times in this document I suggest you perform an internet search. This is perfectly fine. Any other information from the internet or any other source must be documented in the form of comments or in your readme file. A violation of this or working with another student could result in a violation of academic integrity charge.

In chapter 1 we discussed the fact that text in our programs are basically represented using ascii values (0-255 which is 1byte) which is further converted to binary. We saw a table that showed a hello world program in ascii.



I thought it would be interesting to write a program that will read a file, such as helloWorld.c, and write to output the ascii, and binary representation of the file I will discuss the format of your output a little later in the document.

There are many ways to complete this assignment, but to force you to use several new functions/concepts in “C”, I am going to be very specific on much of the instructions. I am aware that some of what I am going to require you to do could be done easier, and maybe even more efficiently. However, as stated above it is my intention to introduce you to several concepts that some of you may not be familiar with. I strongly suggest you read the entire document and make sure you understand completely what you are required to do. **Points will be deducted for not following directions.**

I know most of you know how to read and write information using scanf/printf or fscanf/fprintf. For this assignment you are going to use fread to read the appropriate data from a file. You may or may not have already been introduced to the fread function.

The fread function is used to read binary data. This does not mean the data is 1’s and 0’s. It is just an encoding that may not be human readable. It may be an encoding that is outside of the ascii encoding. There are many text encodings but we will not get into that. The fread function can read an entire chunk of data at one time. There is also an fwrite function that we will not use in this assignment, but it also can write, to a file, a larger chunk of data at one time. For an example of fread and fwrite see the code I gave you in Lab 6.

The syntax for fread is:

**size\_t fread(void\* ptr, size\_t sz, size\_t count, FILE\* fp);**

Where:

**ptr** is the beginning address to the memory that will hold the data being read from the file

**sz** is the size in bytes , of each element to be read

**count** is the number of elements, each one with a size of sz bytes

**fp** is the file pointer that points to the file being read

The return value of fread is the total number of elements successfully read. If the value returned is different than count either a reading error occurred or the end-of-file was reached while reading.

There are many websites that give information about fread. Feel free to google fread and fwrite examples.

**Files:**

You are required to provide three files:

driver.c – This file is where main will be written.

functions.h – This file will provide the function prototypes. You must also provide a header guard in this file. All of the #include files should be included here. You should include the functions.h file in driver.c, as well as, functions.c.

functions.c - This is where you will implement the functions.

**Functions:**

Below I will discuss the functions you will implement for this assignment. Failure to implement any of these functions as described will result in a substantial point deduction.

**void checkArg(int);** - This function ensures the number of command line arguments is appropriate. If this fails you must print to **stderr** a message indicating there were not enough command line arguments and **exit** the program. **FYI: stderr prints to the terminal.**

An example of the command line arguments would be: the executable, input file name, output file name. (./a.out input.txt output.txt) I will test your program with various input files.

**void checkArg(int)** - This function determines whether the file opened successfully. If the file does not open successfully you must print to **stderr** a statement that prints the following:

fopen() failed in file (a file name is printed here) at line #(the line number is printed here), of function (the name of the function is printed here). You are probably wondering what is the relevance of printing the file name, line number, and function name. In this case, none actually. This is just one of those times I want you to learn a specific concept.

So, how do you do this. C provides something called macros.

“A macro is a fragment of code which has been given a name. Whenever the name is used, it is replaced by the contents of the macro. There are two kinds of macros. They differ mostly in what they look like when they are used. Object-like macros resemble data objects when used, function-like macros resemble function calls. ” <https://gcc.gnu.org/onlinedocs/cpp/Macros.html> Macro’s are handled by the pre-processing step of the compilation system. The C programming language has provided macros that will print the current file name, the current line number, and the current function name. You should google something along the lines of “Macros in C”. There are examples that will demonstrate how to do this.

**int getFileCount(FILE\*);** - Because you will use fread when reading the information from the file you need to know the number of characters in the file. This is the purpose of this function. Basically, this function will count the number of characters including spaces in the file. To do this you will need to read each character and increment a counter. There is one additional step that I will discuss in a moment.

**void retFP(FILE\* fp);** - To explain this function, I need to make sure you understand what happens when you use a file pointer to read the content of a file. Remember, a file pointer is simply a pointer. When you call fopen this prepares the file pointer and points it to the beginning of the file being opened. As you read each character, the pointer is incremented to point to the next character. Now let’s consider the function getFileCount again. Remember you read each character in the file using the file pointer. Therefore, after you have read each character, where do you think the file pointer is pointing? If you answered at the end of the file you would be correct. Therefore, we need to have a way move the file pointer back to the beginning, and, **no,** the answer is **not** to call fopen again. C provides a function called fseek. This function allows you to move the file pointer to various places in the file. Search for fseek and you should find the information you need to reset the file pointer back to the beginning of the file. In your research you should have discovered that fseek has a return value. Using this value you must check if fseek was successful. If it was not, then you must print a message to **stderr** that says “fseek failed” and exit the program. If the fseek was successful you must print to **stderr** the position the pointer is currently pointing. This can be done using another function called ftell. Ftell takes one parameter which is the file pointer. Using ftell, print the position of the pointer before calling fseek, then again after you have determined fseek was successful. Now, where do you think the funtionc retFP is going to be called? Hint: At the end of getFileCount. I also want you to call this function in another function, I will discuss that when I get to it.

**void readFile(char\*, FILE\*, int);** - This function will be used to read all of the characters in the file and store them in an array of characters. You must use fread to read all of the information in the input file. Each variable passed to readFile will be used in the fread function call. The char\* is the array that you must store the information when you read the file, the FILE\* is the file pointer that is pointing to the beginning of the file to be read, lastly, int is the variable that holds the number of characters in the file that will be read. Again, in your research of fread, you should have noticed fread has a return value. Using this information, you must check if fread was successful. If it was not successful you must print “fread failed” and exit the program. If fread was successful you must call retFP to return the file pointer to the beginning of the file it is pointing to.

**void ASCII(char\*, FILE\*, int);** - This function will be called to print, to a file, the ascii value of each character in the file. The char\* is the array that holds each character in the file. The FILE pointer is the pointer to the output file, and the int is the count of characters in the file. The format for the output is shown below. Your output format must match the given output’s format.

**void Binary(char\*, FILE\*, int);** - This function will be called to print, to the output file, the binary representation of each character in the file. The format of the output must match the format shown below. You have already written a function that could be adapted to this program. However, you are **not** **allowed to use % and / in this function. You must use bit wise operators such as, &, |, >>, <<, ~, ^.**

The ascii and binary output must both be printed to the same output file, ascii first then binary, as shown below.

**void print(void (\*fp)(char\*, FILE\*, int), char\*, FILE\*, int);** - This function will be used to call the ascii and binary functions. Notice the parameter is a function pointer. This is a simple example of a callback function.

**driver.c**

You will implement the driver.c file. Below are the steps you must take in the driver.

**Steps for the driver:**

Check the number of arguments on the command line.

Create and open the input and output file pointers, call the function to check the success of the opening of the file pointers.

Get the count of characters in the file.

Dynamically allocate the memory for the array of characters that will hold the characters of the file.

Read the file, storing each character, including spaces, in the array.

Create the function pointer that will be used to print the ascii and binary representations of the file.

Call the print function once for the ascii then for the binary.

Close the file

Free the dynamically allocated memory.

**Extra Credit (5 points)**

This is going to be an easy 5 points of EC.

In addition to printing the file information in ASCII and BINARY print the file in HEX. You are **not** allowed to use the C provided hex specifier for printf. Hint: You may want to adapt the print hexidecimal function from the Mimir assignment to this one. However, remember characters are represented by 8 bits or 1 byte. If you choose to do the extra credit you must also make the print function work with the function pointer in this assignment. You should print 9 bytes, with a space between the bytes, per line and each line must be numbered.

**Formatting, Compiling, and Hand-in Requirements:**

Tar your file **PA2.tar.gz** and submit the file through **hand-in to the folder PA2.** If you are doing the EC you must submit a tarred file called **PA2EC.tar.gz** through **hand-in to the folder PA2EC. PLEASE SUBMIT TO ONLY ONE HAND-IN FOLDER.**

You should submit the following files.

* driver.c
* functions.c
* functions.h
* makefile - your makefile should include a make run and make clean. The input and output files will be defined by the command line.
* Readme that has the following information:
  + problems you encountered with this assignment
  + how you solved the problems
  + your thoughts about the assignment.
* With the exception of the makefile, you should have a header in each of your files that contains the following information. If you neglect to place a header in a file, you will receive a 5-point deduction.

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

\*Your name \*

\*CPSC1020 Summer 2017 \*

\*UserName: \*

\*Instructor: Dr. Yvon Feaster \*

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

* Your program should compile with no warnings and no errors on the School of Computing servers. I will use the School of Computing servers to test your program. If your program fails to compile you will receive a 0 on the assignment. If the program compiles but has warnings, there will be a minimal of a 10-point reduction. (The more warnings you have the higher the deductions.)
* Your code should be well documented. (comments – see example below) If you do not provide the appropriate documentation a minimal of 5 points will be deducted for each function that is not documented.
* There should be no line of code longer than 80 characters. There will be a minimal of a 5 point deduction for this violation.
* You must use proper and consistent indention. There will be a minimal of a 5 point deduction if your code is not neat with proper and consistent indention.

Failure to do any of the above items may result in a deduction of points for each offense.

Here are some guide lines for documenting the code in this assignment.

Before each function**, in the functions.h file**, 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.

/\* 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  
 \*/

You are required to have this type of comment block before each function.

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:

Bad

if(something) //This is a comment

{

do something;

}

Good

//This is a comment

if(something)

{

do something;

}

SAMPLE OUTPUT:

Your output must be in the format below. There must be a header above the ASCII and Binary data.

Each line of ASCII must represent 9 characters in their decimal value. Each line must be numbered.

Each line of BINARY must represent one character in the form of the 8 bits that represents the ASCII’S decimal value. Print the binary with a space between each bit. Each line must be numbered.

There will be a substantial deduction of points if the format of your output is not as shown below and described above.

ASCII:

1: 72 101 108 108 111 32 87 111 114

2: 108 100 10 84 111 100 97 121 32

3: 105 115 32 97 32 103 111 111 100

4: 32 100 97 121 46

Binary:

1: 0 1 0 0 1 0 0 0

2: 0 1 1 0 0 1 0 1

3: 0 1 1 0 1 1 0 0

4: 0 1 1 0 1 1 0 0

5: 0 1 1 0 1 1 1 1

6: 0 0 1 0 0 0 0 0

7: 0 1 0 1 0 1 1 1

8: 0 1 1 0 1 1 1 1

9: 0 1 1 1 0 0 1 0

10: 0 1 1 0 1 1 0 0

11: 0 1 1 0 0 1 0 0

12: 0 0 0 0 1 0 1 0

13: 0 1 0 1 0 1 0 0

14: 0 1 1 0 1 1 1 1

15: 0 1 1 0 0 1 0 0

16: 0 1 1 0 0 0 0 1

17: 0 1 1 1 1 0 0 1

18: 0 0 1 0 0 0 0 0

19: 0 1 1 0 1 0 0 1

20: 0 1 1 1 0 0 1 1

21: 0 0 1 0 0 0 0 0

22: 0 1 1 0 0 0 0 1

23: 0 0 1 0 0 0 0 0

24: 0 1 1 0 0 1 1 1

25: 0 1 1 0 1 1 1 1

26: 0 1 1 0 1 1 1 1

27: 0 1 1 0 0 1 0 0

28: 0 0 1 0 0 0 0 0

29: 0 1 1 0 0 1 0 0

30: 0 1 1 0 0 0 0 1

31: 0 1 1 1 1 0 0 1

32: 0 0 1 0 1 1 1 0