Steganography is the art of hiding a message within another form of media: images, articles, books, etc. It is an ancient practice, dating back to approximately 1499. The cool thing about messages encrypted with steganography is that if you don’t know that there’s a message in the image or file, it’s really hard to tell that there is a hidden message! A common example of this is “invisible ink”—unless you know that someone wrote on a paper with invisible ink, then you wouldn’t know to hold it to heat to see the hidden message. Another example can be found here: <http://petitcolas.net/steganography/image_downgrading/>

**ASSIGNMENT OVERVIEW:**

You will implement the functions needed to encode and decode a message in a ppm image.

A PPM image is stored as sequential red, green, blue, values for each pixel. To encode a message, for each character in the image you will strip out the last digit of each color value (slightly changing the colors) and replace the digit with a bit from a binary number. For each character in a message you will convert the character to its binary representation. Then using three pixels you will replace the last digit in the red, green, and blue values with one of the digits from the converted character.

To decode a message you grab the last bit from each red, green, and blue channel in each pixel

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Pixel 1** | | | **Pixel 2** | | | **Pixel 3** | | |
| 255 | 255 | 255 | 122 | 125 | 125 | 73 | 25 | 35 |

001010111

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Pixel 1** | | | **Pixel 2** | | | **Pixel 3** | | |
| 250 | 250 | 251 | 120 | 121 | 120 | 71 | 21 | 31 |

The above binary number translates to 87 which on the ascii table is W

**LEARNING OBJECTIVES:**

This assignment will give you practice working with the following concepts:

* Multiple files
* Command-line arguments
* Pointers
* Structs
* C – I/O concepts (fprintf, sprintf, FILE\*, etc.)
* Dynamically allocating memory
* Reading and writing images
* Many other concepts

**ACADEMIC INTEGRITY:**

This is an individual assignment. You may not receive help from anyone other than myself and the lab TA. Please review the academic integrity policy provided in the syllabus.

**Requirements:**

For this assignments you will have at a minimal eight files:

1. driver.c – This is where **main** lives.
2. ppm.h – I will provide this file. It will contain the prototypes of the functions I implement. These functions deal specifically with a ppm file. It will also provide the definition of two structs.
   1. One for the header.
   2. One for the pixels
3. ppm.c – This file is where I implemented the functions listed in the ppm.h file.
4. EncodeDecode.h – I will provide this file. It will contain additional prototypes of functions I needed to encode and decode a message in an image.
5. EncodeDecode.c – This file is where I implement the functions listed in EncodeDecode.h
6. utils.h – this provides the prototypes for two functions needed to check the number of command line arguments passed to the program. It also contains a function that check if the file pointers opened successfully.
7. utils.c – this file is where I implement the functions for utils.h
8. makefile

Each of your header files (.h) must have header guards. Points will be deducted if you do not provide header guards I also removed all of the #include statements, you will need to provide these.

Below are the prototypes and explanation of some of the functions for this program.

**header\_t readHeader(FILE \*);**

This function reads the header information of a ppm file.

**pixel\_t\*\* readPixel(FILE \*, header\_t);**

This function reads the values of the pixels in the ppm image. Notice this function has a return value. It is in this function, that you will dynamically allocate the memory for the 2D array. We discussed several versions of allocating memory for a 2D array. You may use any of these methods **EXCEPT** the one that allocates memory for the pointers on the stack. If you are not sure about which method not to use, you should come see me. You will then read in the pixels from the image storing the pixels in the 2D array you just created and return the 2D array.

**void writeHeader(FILE \*, header\_t);**

This function will be used to write the header to the output ppm file. Use fprintf to write the header information to the output file.

**void writePixels(FILE \*, pixel\_t \*\*, header\_t);**

This function will be used to write the pixels to the output ppm file. Use fprintf to write the pixels to the output file.

**void removeDigit(pixel\_t\*\*, header\_t);**

This function will be used to remove the value of the ones place of each channel in the pixel and replace it with 0. (What???) Consider the following:

If the red channel of the first pixel has an integer value of 255, you need to reduce the value by 5 to equal 250.

**void encodeMsg(FILE\* in, FILE\* out, char\* msg)**

As shown in the above image, for each character, this function should add a 1 or 0 to the value of each of the three channels for the three pixels. Below are the steps I took in my function.

1. Create the local variables needed for this function. This included a couple counters, one for the character counter, which came in handy when accessing the char \* msg, which is holding the literal string that represents the message you will encode. One for the bit counter. I also have an int array to hold the 9 values that will be added to the 3 RGB values for the 3 pixels.
2. Call writeHeader.
3. For each character in the message convert the character to it’s binary number equivalent. (charToBinary function).
4. Encode the binary number in the image.
5. After you have looped through all the pixels and there are no more characters to encode write the new pixels to the output file by calling writePixels.

You will need nested loops and some if statements for this function.

**void decodeMsg(FILE\* in);**

There are several steps needed to decode the message.

1. Since this is an ppm file that the message is encoded in you first should call readHeader to read the header of the encoded ppm image.
2. Allocate the memory for the pixels of the encoded ppm image. Suggestion: you could create a function that allocates the memory for the pixels then return the double pointer.
3. Call readPixel to read the pixels of the encoded ppm image storing them in the memory you just allocated.
4. Grab the last digit of the red, green, and blue numbers. When you grab these digits save them as integers. You should have an integer variable each for red, green, and blue.
5. For each number you grab send it to the queue function.
6. You will continue this process for all pixels in the image. As an example, for the first red digit I grab I will call queue passing in the digit, an array that will be used by queue to store this digit and the address of a variable that will be used to keep track of the number of digits passed to queue.

**void queue(int n, int bin[], int \*index);**

This function is called by decodeMsg. As the decodeMsg function grabs the ones digit of each pixel channel (red,green, and blue) it passes that value to queue as “n”. This function keeps track of the “n’s” passed to the function. Notice bin is an array which should be of size 9. An array created in decodeMsg is passed to queue, a.k.a. bin. Also, “index” is a pointer that is pointing to a variable passed to queue by decodeMsg. The queue function will continue to add the digit represented by “n” to bin until 9 digits have been added. Once 9 digits (1 or 0) are received you now have the binary bits needed to convert the content of bin to a character by calling binToCharacter. Obviously, the character returned from binToCharacter should be less than 256. If the value retuned from binToCharacter is less than 256 then queue prints the character.

**void dec2bin(int\* , int);**

This function converts a decimal number to binary storing the bits in an array.

**unsigned char binToCharacter(int\* input);**

This function is used to convert a binary number to decimal. Notice this is returned as an unsigned char. Remember we are working with three pixels which equals 9 bits but the most significant bit is always 0. Why???

**void checkArgs(int, int);**

This function checks that the appropriate number of command line arguments were used when running the program. If the appropriate number of command line arguments are not passed to the command line you should print a message to stderr that tells the user that there were not enough command line arguments and exit the program.

**void checkFile(FILE\*);**

This function should check if the file pointer opened successfully. Basically, check if the file pointer is NULL. If it is NULL, print to stderr a message that states the file did not open correctly and exit the program.

**driver.c file**

I will give you my driver file. This does not mean you have to use this file. However, your driver file should have a minimal amount of code in it. Points will be deducted if you have an excessive amount of code in main. If you have code that can be placed in a function then it should not be in main. Write a function and call that function in main.

**You are not required to use this outline for your program. I love seeing the many ways students approach and solve a programming problem. However, you may not put everything in one file. You should have multiple functions that for the most part do small task. I will deduct points if you have large functions. I will also deduct points if you do not have the minimal 8 files listed above. You may change the function prototypes, this will require you to change the driver file as well. The description above should serve as a guide for you but you may develop your own solution. The things you must do is allocate memory, free any allocated memory. Use file pointers**

**Makefile:**

You are required to provide a makefile. Your makefile should have a make run. In addition to the executable, there will be two ppm files as command line arguments. The target for run, in your makefile, should name these files **input.ppm** and **output.ppm**. In otherwards, when I type the command - make run - your makefile should compile the program then run the program using the executable and two files named **input.ppm output.ppm.** I will run a script to add the two files that I plan to use to your folder. Therefore, you should **NOT** provide ppm files named input.ppm nor output.ppm when submitting. If you have nested files you need to make sure the makefile is in the top most folder. Points will be deducted if your make run does not work due to nested folders or the makefile being in the wrong place.

**Testing your program:**

I will provide you with a ppm image that you may use to test your program with.

**FORMATTING:**

You will need to add a header to each of your files like the following:

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

\*Your name

\*CPSC 2310 F21 Section 00?

\*Your email

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

Your program should compile with no warnings and no errors. If your program does not compile the highest grade you can get for the assignment will be 20. If your program compiles but has warnings, there will be a deduction up to 20 points.

* Your code should be well documented. (comments). See comment information below.
* 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 three above will cost a minimum of 5 points. A high number of offenses will increase the number of point reduction.

**HANDIN:**

Use handin.cs.clemson.edu to submit your files. I have created buckets named PA1. If you complete the extra credit list below you should submit your files to PA1EC. **DO NOT SUBMIT TO BOTH PA1 AND PA1EC.**

Things to do prior to handing in your files:

1. **Test your program on the SoC servers**. I will not accept the excuse “It compiled on my computer.” I test programming assignments on the SoC servers. You should test your program with various messages. I will test your program with several messages. I will test your program with not command line arguments as well as with incorrect file names.
2. You should also provide a **README** that consist of the following.

* A short description of any problems you encountered when writing this program.
* How you solved the problems you encountered.
* Your thoughts on the assignment. This is your opportunity to tell me if you like the assignment or not. What you did or did not like about the assignment. Anything you want to tell me.

**If you do not provide a README file, you will receive a 10 point deduction.**

1. Tar zip your files naming the tarred file PA1.tar.gz or PA1EC.tar.gz if you are doing the extra credit.
2. It is your responsibility to make sure you submit all of the appropriate files and that the files are in working order. Hand-in allows you to check your files after submission. You must check your files. If the files are corrupt you should resubmit. This could take time so be sure not to wait to the minute to submit. Any submissions with corrupt or missing files will result in a 0 on the assignment.

Here are some guidelines for documenting the code in your assignment.

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:

Bad

if(something) //This is a comment

{

do something;

}

Good

//This is a comment

if(something)

{

do something;

}

EC: 5 points

In addition to all of the other requirements for this assignment. Write a function that will check for and ignore comments and whitespace in the header of the ppm file. This function should be called each time you read the header of a ppm file.

**FYI: I nor the TA’s will help with EC. EC is something you need to figure out yourself.**