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 interesting fact about steganography it is becoming a popular tool for cybercriminals. Cybercriminals are using it to send messages, as a means to transmit malware, and many other forms of criminal activities. It has become so prevalent that in 2016 an organization called “Criminal use of Information Hiding – CUING.org was formed by the European Cybercrime Center – EC3.

**ASSIGNMENT OVERVIEW:**

You are going to create a program that will encode a message in a ppm image. A PPM image is stored as sequential red, green, blue, values for each pixel. You are going to 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.

The image below represents three pixels. Pixel 1’s RGB values are 255, 255, 255 respectively. Pixel 2’s red value is 122, green is 125, and blue is 125, etc.

A few things you will need to to:

1. Change the last digit in each pixels RGB values to 0 (this will need to be done for all pixels in the image not just the pixels being used to hold the encoded message.)
2. Convert a character, from the message to be encoded, to it’s decimal format
3. Replace the 0’s (from 1 above) in the RGB values with the 8 bit binary number representing the character converted from 2 above.

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **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
* FILE I/O
* 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 must have at a minimal of three files:

1. driver.c – This is where **main** lives.
2. encode.h – This is where you will provide the definition of the structs. You should have two structs.
   1. One for the header.
   2. One for the pixels

This file also will have the function prototypes. The following are the functions I wrote for my encoder.

Your header file should use header guards, #ifndef / #endif in your .h file.

1. encode.c – This file will contain the implementation of the functions associated with this program.

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

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

This function reads the header of a ppm file.

**pixel\_t\*\* allocatedMemory(header\_t);**

This function should dynamically allocate the memory for the 2D array of type pixels\_t that will hold the RGb values for the image.

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

This function reads the values of the pixels from the ppm image. Notice this function has a return value. It is in this function, that you will call allocateMemory for the 2D array. You are required to use the method that allocates the memory first for the pointers that will point to the actually elements in that row. We discussed this method in class. If you are not sure about which method, you should use come see me. You will then read in the data from the image returning the 2D array.

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

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

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

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

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

This function will be use to subtract the value of the ones place of each channel in the pixel. (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. We have access to an operator that will make this easy. What is is? You **MUST** do this step for all pixels in the image.

**void encodeMsg(FILE \*, header\_t, pixel\_t \*\*, char \*);**

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 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.
4. Encode the binary number in the image described above.
5. After you have completed looping through all the pixels and there are no more characters to encode write the new pixels to the out file using writePixels.

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

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

This function should be the same as the function you wrote in lab, with the exception of the size of the array is 9 rather than the 8 you use in lab.

**driver.c file**

The driver should have a minimal amount of code in it. You must check that the user used the appropriate number of command line arguments. Create and open the input and output file pointers, you must ensure they open appropriately.

In addition to the file pointers, there are various variables you will need for this program. Some I used are as follows:

1. A variable for the header information.
2. A double pointer variable of type pixel\_t that will hold the RGB values for the image.
3. A char pointer for the string literal that will represent the message you will encode.

You will need to call readPixel which will return the dynamically allocated memory address to the double pointer created above.

Now that you have the allocated pixel and a header with the image header information stored in it, you should call the removeDigit function.

You now have everything you need to call encodeMsg.

As discussed in class 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, also as stated in class you may not put everything in one file. You should have multiple functions that for the most part that do small task. I will deduct points if you have large functions. You will also deduct points if you do not have the minimal three files listed above.

**Testing your program:**

I will provide you with an executable to use for testing. Here is the how you use this executable: ./key < *the name of the encoded ppm file*

Notice this uses redirection.

When you run this executable the message you encoded in the image should appear on your screen.

**FORMATTING:**

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

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

\*Your name

\*CPSC 1020 Sm19

\*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)
* There should be no lines of code longer than 80 characters.
* You should use proper and consistent indention.

**HANDIN:**

Use handin.cs.clemson.edu to submit your files. I have created buckets named PA1.

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.
2. Tar zip your files naming the tarred file PA1.tar.gz. Your tar file should **NOT** have nested files. Change directories to the folder that contains your file, using a terminal type tar –cvzf **PA1.tar.gz** \* The (\*) is a wildcard and tells the tar utility to tar everything in the folder.
3. 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.

1. 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.
2. Your code should be well documented. Below are some guidelines on how to comment your code. You must have detailed documentation for each function in the encode.h file. When implementing the programs functions you should also have comments within the code to explain what your algorithm is doing. You do not need to comment each line nor code that is obvious. Code that is not obvious you will need to explain.

Here are some guide lines 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.

/\* 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 not required to do yours exactly like this one. However you must provide this type of information.

Also, if you include comments in the body of the function (and you should) they must 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;

}

**Extra Credit: 10 points**

In addition to the encoder, write a decoder that will decode the message hidden in an image.

You should have a separate set of files for the decoder. (decoder.c, decoder.h, driverEC.c) This will allow me to test your decoder separate from your encoder. You can tar.gz the files together.

If you choose to do the EC submit your tar.gz file to the PA1EC folder only. The formatting requirements above apply to the EC.