CIS 2107 Assignment 6
Steganography Assignment

Description

In this assignment, you'll write two short programs: one which hides a short, secret message in an image file, and a second program which reads the hidden message back out of the image.

In order to keep the program simple, your program will only hide short messages in PPM files (a simple, uncompressed format). Hidden messages are a maximum of 255 characters (if this seems overly restrictive, let's just consider it the Twitter of stego programs).

As a sample, two files are provided, TU.ppm, and TU_modified.ppm, both of which look identical, and both have exactly the same file size, however the 2nd file contains the hidden text "Super secret message. Spooky".

Image File Format

A PPM file has the following format:

- A "magic number", specifying the file type. PPM files have magic number "P6" (in ASCII)
- Whitespace (this could be a blank, tab, carriage return, line feed, …)
- Optional comments. A comment is a line which begins with #. Your program may safely ignore all comments.
- The width in pixels of the image in ASCII.
- Whitespace
- The height in pixels of the image in ASCII.
- Whitespace
- The maximum color value in ASCII. Your program need only handle the value "255".
- A single whitespace character.
- Pixel data. You need only handle files with a maximum color value of 255, so each pixel will be represented by three bytes, one byte each for red, green, and blue. If your image is X rows by Y columns, then the file will contain X*Y*3 bytes of data, plus a header.

WriteMsg

This program should take the following as command-line arguments:

- A secret message, given as a string.
- The name of an image file.
- The name to give the output file, i.e., the image file containing the secret message.

Basic operation

WriteMsg should:

- Read the PPM header
    - if the file does not begin with the text "P6", print an error message and quit.
    - Read and ignore comments, if any.
    - Read the number of rows and colums of pixels. If there are an insufficient number of bits in the file in order to hide the message specified, print an error message and quit.
    - Read the maximum color value. If this is not 255, you may print an error message and quit. Otherwise, we can assume that each pixel of the file is represented as three one-byte values, corresponding to the level of red, green, and blue of the pixel.
- Hide a message in the PPM data.

    As we discussed in class, you're going to hide your message in the least significant bits of pixel data in the image. So, for example, if we're going to hide a particular byte b in the image, the 8 bits of b will be spread across the least significant bits of 8 bytes of the image file (i.e., hidden in the data for 2 2/3 pixels).

    So that the message reading program knows how long the hidden message is, and knows when to stop reading, before hiding the bytes of the message itself, hide the message length.

    If you're unclear about how to hide the message, think of the lab where you implemented the arithmetic operations using bit operators. What you need to do here isn't nearly as tricky.

ReadMsg

This program should take a single command-line argument, the name of the file containing the hidden message.

ReadMsg should read the PPM header, then hidden in the least-significant bits of the pixel data, it should read the length of the message, which is hidden in a single byte spread across the first 8 data bytes of the picture. It should then read the hidden message itself and print it to the screen.

Some Suggestions

Organization

You may find it helpful to organize your programs into four files:

- **stego.h** contains declarations of constants, structs, functions, etc., common to both programs

- **stego.c** the implementation of functions common to both programs
- **WriteMsg.c** the message hiding program
- **ReadMsg.c** the message reading program

You might also find it useful to implement a function like:

```
/* Obtains the next hidden byte from the stream referenced by fp */
/* returned as an unsigned char converted to an int. If the stream */
/* is at end-of-file or a read error occurs, returns EOF */
int read_hidden_byte(FILE *fp)
```

In other words, the function should work exactly like fgetc()/getchar(), except that it reads and return hidden bytes of the secret message. The read program would call this once to determine the length of the hidden message, and then in a loop to read the message itself.

Similarly, you could also write a function like:

```
void write_hidden_byte(char c, FILE *fp)
```

which would write the byte c to the stream referenced by fp, with the bits of c hidden in the low-order bits of the next 8 bytes of pixel data, as described elsewhere in the assignment. You'd call this function once to write length of the secret message to the file, and then again in a loop to hide the message.

Tools and other resources

- gdb is your friend. Get to know it.
- hexdump can be very helpful. Use it to examine the image file without the hidden message and then the corresponding image file including the secret message. For example, tu.ppm.txt is what the TU file looks like after running it through hexdump.
- A completely white640_480.ppm file for you to use for testing. Once you've written your program and you're trying to figure out why it's not working, you'll understand why this could come in handy.
- Gimp is a free graphics editor; it's been ported to most popular operating systems. You can use this to create and view ppm files. Maybe it'll also come in handy some time in the future if you don't feel like shelling out the money for Photoshop.
- More information on the ppm format can be found on the netpbm sourceforge page. Note that there are several other file formats discussed on the page (including a version of PPM which stores the image data in ASCII and not binary. These have a magic number of P3, and not P6. Please don't confuse yourself by reading about the wrong file format).

What to submit

Send a zip file containing all of your source files along with a README file containing a road map for your code, along with instructions for how to build it.

As usual, your program should be well documented. To make things easier for your lab instructor, please be sure to include your name at the top of all of your source files, place your code in a directory named something like Stego_YourLastName, which itself will be inside a zipfile called Stego_YourLastName.zip