**CSC221: Computer Organization and Assembly Language**

**Spring 2017**

**Project: Steganography**

Images are digitally represented as a sequence of picture elements, known as pixels, arranged in a series of rows and columns. Several file formats (e.g. jpg, bmp, png etc.) have been created over the years to store images for later viewing. One such format is the PPM file format. The PPM file format provides a straightforward way of accessing image data for manipulation. A PPM image file has the following format:

* A "magic number" for identifying the file type. A ppm image's magic number is the two characters "P6".
* Whitespace (blanks, TABs, LFs).
* A width, in decimal.
* Whitespace.
* A height, again in decimal.
* Whitespace.
* The maximum color value (Maxval), again in decimal. Must be less than 65536 and more than zero.
* A single whitespace character (usually a newline).
* Rows of pixel values, in order from top to bottom, where number of rows are equal to *height*, Each row consists of pixels equal to *width*, in order from left to right. Each pixel is a triplet of red, green, and blue samples, in that order. Each sample is represented in pure binary by either 1 or 2 bytes. If the Maxval is less than 256, it is 1 byte. Otherwise, it is 2 bytes and the most significant byte is first.

See the attached sample images. More information on the PPM file format can be found at <http://netpbm.sourceforge.net/doc/ppm.html>

**Steganography**

Steganography is simply the hiding of information within image data. In this project, you are to write a program in assembly language that allows a user to “hide” a string of arbitrary length within your image data. How is this to be accomplished? Through bitwise manipulation.

Consider the following image:



|  |  |  |
| --- | --- | --- |
| R:246 | G:189 | B:134 |

The color of the pixel highlighted in the image above is represented using the RGB color model. In this model, three separate colors (Red, Green, and Blue) are blended together to create the final color of the pixel. The PPM images that we will work with in class will limit the range of values that can be represented in each color channel to 256 different values (0-255). Thus, each pixel of your PPM image data can be represented using three (3) characters. Considering the pixel in the image above as a sequence of bits, we arrive at the following:

|  |  |  |
| --- | --- | --- |
| 11110110(246) | 10111101(189) | 10000110(134) |
| Red Channel | Green Channel | Blue Channel |

Now if we were to alter the least significant bit (LSB) of each channel value, we would arrive at the following values:

|  |  |  |
| --- | --- | --- |
| 1111011***1***(247) | 1011110***0***(188) | 1000011***1***(135) |
| Red Channel | Green Channel | Blue Channel |

Although the numeric values are different, the effect on the color of the pixel is negligible. The human eye will not be able to detect the slight change in color.

Now consider the binary representation of the ASCII code for uppercase letter ‘Q’.

|  |
| --- |
| 0***101***0001 (81) |
| The binary representation of ‘Q’ |

Notice that bits 5, 6, and 7 contain the same values as the LSBs that were modified in the pixel above. In fact, bits 5, 6, and 7 of the letter ‘Q’ have been scattered across the three individual channels of the pixel. Now imagine spreading the bits of an entire phrase or document across multiple pixels and color channels in an entire image. Doing so would effectively hide your data in an image without any adverse effects on the image itself.

In this project, your program that will first read a PPM image and store in an array. You can assume that each color channel will contain a maximum value of 255. Therefore, you can use three (3) bytes to represent a single pixel. The next step is to hide a user provided phrase in the image. In your program, you will prompt the user for a phrase to be hidden inside of a PPM image. You can assume that the phrase will be short enough to store within the dimensions of the PPM image. Be sure to add the null terminating character to the end of the string that you wish to hide. This character will be of extreme importance during the decoding process. Once the data has been hidden, you should write your image data to a new PPM image file. Be sure to not overwrite the original PPM image. Your program should also allow the user to extract hidden messages from PPM images. To do so, you should reverse the process used to hide the data. Extract the LSB from every color channel to construct 8-bit characters. When you extract a null terminating character you should stop searching for additional data and report the message that you have uncovered.

Sample Program Execution:

What would you like to do?

1.) Hide

2.) Recover

Enter your selection: 1

Please specify the source PPM filename: 1.ppm

Please specify the output PPM filename: hidden-1.ppm

Please enter a phrase to hide: The rooster crows at midnight

Your message “The rooster crows at midnight” has been hidden in file: hidden1.ppm

What would you like to do?

1.) Hide

2.) Recover

Enter your selection: 2

Please specify the source PPM filename: hidden-1.ppm

The following message has been recovered from file hidden-1.ppm: “The rooster crows at midnight”.

Deadline: June 22, 2017.