MTRX1702 - C Programming
Assignment 2

This assignment requires you to design and build a program that hides the contents of a data file inside a bitmap file. The original data file may subsequently be recovered from the modified bitmap file.

This assignment should take an average student 12 hours to complete.

Submission Deadline: 23:59 pm on the Sunday, 27th of October.

Late submissions will be penalised.

The proportion of the overall marks allocated to each component of the assignment is indicated after the title enclosed in square brackets.

1 Background

Steganography (which means "concealed writing") is the art of writing hidden messages, disguised as a clearly visible but innocuous cover message.[1] Historically, hidden messages have taken many forms including the use of invisible ink, or marking selected letters with pin-pricks, or using cut-out grilles to cover most of a message except the relevant letters. In ancient times, Histiaeus shaved the head of a trusted slave, tattooed a message on it, and hid it by waiting for the slave's hair to regrow. In the period following the first World War, covert messages were photographically reduced to the size of a full-stop and printed as *microdots* in ordinary letters and sent via regular post.

The purpose of steganography is to send a message without attracting the attention of unwary third parties. This is different to encryption, where others may be aware of and possibly intercept encoded messages, but are unable to interpret them. Even with unbreakable encryption, an encoded message is likely to arouse suspicion, which is problematic for a covert or under-cover operative. Steganography on the other hand permits subterfuge in plain sight of the enemy, allowing the obscured message to pass by unnoticed.

The current digital age facilitates a form of steganography wherein the low-order bits of an image or audio file are replaced with the message bits. These bits typically have insignificant influence on the appearance or sound of the original file, and so they can be modified without noticeable effect. This assignment is concerned with image-based steganography. You are to hide the contents of a data file in the low-order bits of a bitmap image file, and later retrieve the original data from a modified image.

2 Windows Bitmap Files

Your program is to operate on uncompressed Windows Bitmap files with 24-bit colour encoding. This is one of the simplest available file-formats and is composed of a "header block" followed by colour data for each image pixel.

```
[header info] [pixel 1] ... [pixel N]
```

[1]For more information on Steganography, see `http://en.wikipedia.org/wiki/Steganography`.

The contents of the header is not relevant to this project. All you need to know is how many bytes are in the header so as to skip over them to get to the pixel data. Each pixel is represented by three bytes; one each for the red, green and blue components of its colour. Thus, the intensity for a component (eg., red) of a single pixel is given by an 8-bit value (ie., 0 to 255 `unsigned char`). It is therefore clear that the low-order pixel information is simply the least-significant bits of each byte.[2]

Your program shall copy the header block of the bitmap to the output file directly and without change. The hidden message is to be stored in the low order bits of the pixel data. You will be provided with code that analyses the bitmap file and tells you the number of bytes in the header and the number of bytes of pixel data, so that the separate treatment of header and pixel data is trivial.

3 Basic Specification [60%]

Interface. The program is to have a command-line interface. This means using the alternative form of `main()` that takes command-line arguments (see Section 13.4 of the course text).

```
    int main(int argc, char **argv)
```

The interface for encoding a data file inside a bitmap file takes three filenames as arguments.[3]

```
    steg <bmpfile> <datafile> <outputfile>
```

In this case, `<bmpfile>` is a Windows bitmap file, `<datafile>` is the data file, and `<outputfile>` is the modified bitmap that contains the hidden message. The interface for decoding a modified bitmap and recovering the data file takes only two arguments.

```
    steg <bmpfile> <outputfile>
```

Here `<bmpfile>` is a modified bitmap and `<outputfile>` is the recovered data file.

Encoding algorithm. To store the data file inside the bitmap file, the following steps are carried out.

- The BitmapOffset field of the file header is read to identify the start of the pixel data.
- The header block is written to output verbatim.
- The first 32 bytes of pixel data are reserved to store the size of the input data file. The size is to be stored in the least-significant bit of these 32 bytes.

[2]Compare this to the 8-bit Windows Bitmap file-format, where each pixel is represented by a single byte, and colour is determined by a colour-map lookup table. In this case the low-order information is not necessarily the least-significant bits of the byte.

[3]Here the executable is named `steg`. You may give your executable a different name, if you wish. It does not matter since you will only submit your source code files.

- Compute the amount of space required to store the input data. From this, compute the number of bits in each pixel byte that will have to be modified so as to store the data. For example, a small data file of `N` bytes might fit into the least-significant bit of the first `8N` pixel bytes. A larger file might need to modify the two lowest bits of some bytes. In the worst case, the data file will modify all 8 bits of some or all of the pixel bytes, (in which case the data will not be well hidden).
- Store the bits of the data file in the low-order positions of the pixel bytes.

The last step requires the use of bitwise operations. You will read in a byte from the data file and spread its bits across several pixel bytes.

Decoding algorithm. To extract the data file from the modified bitmap involves the following steps.

- Skip over the bitmap header block.
- Get the number of bytes of the data file from the first 32 bytes of the pixel data.
- Compute the number of bits from each pixel that store the hidden data.
- Extract the relevant bits and reconstruct the data file.

Messages. The program shall provide feedback to the user by printing messages to the screen. There are three non-error messages.

- Usage message. If the program is run with no additional arguments, or the wrong number of arguments, it is to print the following message and terminate.

```
Usage: (encode or decode, respectively)
    steg <bmpfile> <datafile> <outputfile>
    steg <bmpfile> <outputfile>
```

- Overwrite output file. If the output file already exists, query whether to overwrite, terminating the program if the user doesn't type `'y'`.

```
Output file <filename> already exists. Overwrite (y/n)?
```

- Maximum number of bits modified. The ratio of the data file size to the number of pixel bytes will determine the maximum number of low-order bits per byte that are overwritten. The program shall print this number.

```
There was a maximum of <N> bits modified per byte.
```

Errors. If the program encounters an error, it is to print an error message to the screen and terminate. In particular, the program shall include the following error messages.

- Unable to open a file.

  ```
  Error: Could not open file <filename>.
  ```

- Encoding error, where the bitmap is not big enough to store the entire data-file.

  ```
  Error: Bitmap too small to store data file.
  ```

- Decoding error, where the 32-bit value that denotes the size of the hidden data is larger than the available space in the bitmap.

  ```
  Error: Expected data size is larger than available space in bitmap.
  ```

3.1 Compiling

Your submission will be compiled using the following command in the Visual Studio 2010 Command Prompt:

```
cl *.c
```

If your program does not compile when this command is run, it cannot be assessed for compliance with the specification and you will receive 0% for this component of the assignment.

4 Extensions [20%]

If the basic specification as described above is implemented, the maximum possible mark for the assignment shall be 80%. In order to gain a mark of greater than 80%, one or more extensions must be implemented. These extensions should extend the functionality of the above program in a useful and appropriate way, given the specified goals of the program. Some suggested extensions are:

- Include an interactive user interface in addition to the command-line interface.
- Calculate a checksum and include that checksum within the hidden message to validate the successful decoding of a message.
- Calculate an indication of the overall level of file corruption caused by the steganographic process.
- Implement compression by identifying whether the input file uses only ASCII characters. (This would require additional 'header' information beyond the size of the output file to be included in the hidden data.)
- Parse the header to verify that the file is actually a supported bitmap.

- Implement Doxygen-style commenting to enhance the quality of the project documentation.
- Create a makefile so that the program can be compiled by gcc using the single command *make*.

Other possible extensions will also be considered. The suitability of any other extensions should be discussed by e-mail with the Lecturer (j.ward@acfr.usyd.edu.au).

5 Documentation [20%]

The documentation for this assignment shall comprise two components. In-source documentation, and a separate short report.

Appropriate commenting and descriptive variable names should be used in all source code to maximise readability. This will be marked based on the ability of the reader to understand the operation of the code without reference to the accompanying report. This in-code documentation should NOT discuss design decisions, but simply the implementation of the program. Each function must have a comment header block describing the inputs, outputs and a one-sentence description of the operation of the function.

The short report shall contain a discussion of each module within the program, the functionality of the module, and the data-flows between each of the modules. Do not discuss each individual function within the program. Simply discuss the functionality of each module as a whole, their dependencies, and list the component functions. If a multi-file approach to the project has been implemented, ensure the file boundaries are indicated within the modular discussion. Use diagrams where appropriate. This report is expected to fill 2-3 pages.

The report shall also include a "Statement of Compliance", which describes how well the implemented programme complies with the specification in the assignment sheet. It must identify all points of non-compliance, and may provide a justification for this non-compliance. The Statement of Compliance must also clearly and briefly describe the additional functionality provided by any extensions.