

COSC 515: Security

Homework #1

For this assignment, you will write an application that will allow for least significant bit steganography to be used to store a string of text encoded with ASCII in a bitmap image. To achieve this, your application will provide the user with options to embed the text into an image and an option to extract the text from the image.

The workflow for embedding the text should be the following:

1. Place a bitmap image named `original.bmp` in the working directory of your project. Your code should open this image for use.
2. Prompt the user to enter a string.
 - The string entered by the user must be valid ASCII input and have a length greater than 0.
 - If either of the above conditions are not met, the user should be prompted repeatedly until they provide input that meets these conditions.
3. Determine the number of bits that are available in the image as well as the number of bits that are needed for storing the string.
 - If the number of bits needed for storing the string is less than or equal to the number of available bits, then write the string to the bits.
 - If the number of bits needed for the string exceeds the number of available bits in the image, then determine what portion of the string can be written to the image, print a warning to the terminal that shows what portion of the string will be written, and then write the truncated version of the string to the bits that are available in the image.
4. Write the modified version of the image data to a new image file named `steg.bmp`. The bits should be written to the image starting with the top left pixel, then moving right, and advancing to the first pixel of the next row when necessary. For each pixel, there are three bytes—one for each color channel (red, green, and blue). You should write to these color channels in this order.

The following diagram depicts an example of the order in which writing to the least significant bits of a 3×3 image should take place. Each square represents one pixel/three bytes.

Bits 1-3 Red: 1 Green: 2 Blue: 3	Bits 4-6 Red: 4 Green: 5 Blue: 6	Bits 7-9 Red: 7 Green: 8 Blue: 9
Bits 10-12 Red: 10 Green: 11 Blue: 12	Bits 13-15 Red: 13 Green: 14 Blue: 15	Bits 16-18 Red: 16 Green: 17 Blue: 18
Bits 19-21 Red: 19 Green: 20 Blue: 21	Bits 22-24 Red: 22 Green: 23 Blue: 24	Bits 25-27 Red: 25 Green: 26 Blue: 27

The workflow for extracting the text should be the following:

1. Open the `steg.bmp` image.
2. Reassemble the least significant bits back into the original string by iterating over the image's pixels and color channels. Once the null terminating character is found, the iteration should be stopped.
 - An edge case that I will check for when running your application is how it handles a scenario in which an attempt to extract a string is made but no null terminating character is found. If this happens, the application should print a warning to the user stating that no null terminator was found which means that the image probably doesn't have an embedded string, and thus, the output string is likely arbitrary nonsense.
3. Print the reassembled string to the terminal.

When writing your application, here are a few important things to keep in mind:

- Strings have their ending marked with a null character. You should account for this when writing the logic for storing your data in the image's bits so that when you retrieve your string from the image, you can determine where the end of the string is. For example, if your input image only has enough bits to store 3 characters, and your input text is `Hey`, the input must be truncated to `He` since the third character that will need to be stored will be the null terminator.
 - For any images that are too small to hold a single character, an error should be printed to the terminal alerting the user of this, and `steg.bmp` should not be outputted.
- Each ASCII character uses 8 bits. This means that if the number of bits available for data storage in the image is not divisible by 8, then there will be anywhere between 1 to 7 bits that are unusable for storing your data. You must keep this in mind when calculating how many characters can be stored.

- It is important that your application's steganography implementation follow the rules described above. This is because, for testing purposes, your application's input and output must be compatible with my application. If I embed text into an image using your application, I should be able to extract it using my application. Likewise, if I embed text into an image using my application, I should be able to extract it using your application. If your application is not compatible with mine, you will receive a 25-point deduction.
- Although you can and should test your application using your own bitmap images, you should also test using the images contained in `test-images.zip` on Canvas. The images in this archive are the images that I will be using for testing. Please see the `README.txt` files contained within the archive for details on how these images are used.

For this assignment, you are allowed to:

- Write your application in any programming language.
- Use third party libraries to aid in image processing.
 - **WARNING:** Third party libraries that provide tooling for performing steganography are not permitted.
- Use the internet and class resources to figure out how to implement Least Significant Bit steganography.
 - In the case of internet resources, you must list the resources you used in a comment block at the top of your code.
- Ask me questions.

For this assignment, you are not allowed to:

- Copy and paste code from the internet, even with attribution.
 - Being allowed to use the internet for researching does not give you freedom to find code that already does this and submit it as your own!
- Take code from or give your code to other students and/or collaborate to submit one assignment under multiple names.
- Use any third-party libraries or APIs built into your programming language of choice that provide a least significant bit steganography implementation for you.

Grading

This assignment is worth 50 points. 25 points will be based on the accuracy of your implementation of embedding text and the remaining 25 points will be based on the accuracy of your implementation of extracting text. Submissions that do not compile, or in the case of interpreted languages, do not run properly, will be receive a 25/50.

Once you have completed the assignment, create a zip archive file containing all related project files and upload it on Canvas. This assignment is due on the date listed on Canvas.

Application Workflow Examples

This screenshot shows an example of how invalid user input should be handled.

```
Command Prompt - steg.py X + v

(steg) D:\School Stuff\Teaching\Spring 2024\COSC 515\Homework\steg>steg.py
Choose an option:
1. Embed string in bitmap.
2. Extract string from bitmap.

Enter the number corresponding to your desired option: garbage

Your input was invalid. Try again.
Enter the number corresponding to your desired option: 1

Enter the string to embed: 🍌

Your input is either empty or not a valid ASCII string. Try again.
Enter the string to embed:

Your input is either empty or not a valid ASCII string. Try again.
Enter the string to embed: |
```

This screenshot shows an example of embedding the string “This is a test. This is only a test.” into an image that has enough available bits for storage to support the entire string.

```
Command Prompt X + v

(steg) D:\School Stuff\Teaching\Spring 2024\COSC 515\Homework\steg>steg.py
Choose an option:
1. Embed string in bitmap.
2. Extract string from bitmap.

Enter the number corresponding to your desired option: 1

Enter the string to embed: This is a test. This is only a test.

(steg) D:\School Stuff\Teaching\Spring 2024\COSC 515\Homework\steg>
```

This screenshot shows an example of extracting the string from the image that was created in the previous screenshot.

```
Command Prompt

(steg) D:\School Stuff\Teaching\Spring 2024\COSC 515\Homework\steg>steg.py
Choose an option:
1. Embed string in bitmap.
2. Extract string from bitmap.

Enter the number corresponding to your desired option: 2

The following string is embedded in the image:
This is a test. This is only a test.

(steg) D:\School Stuff\Teaching\Spring 2024\COSC 515\Homework\steg>|
```

This screenshot shows an example of embedding a string into an image that does not have enough bits for storing the string, thus resulting in the string getting truncated.

```
Command Prompt

(steg) D:\School Stuff\Teaching\Spring 2024\COSC 515\Homework\steg>steg.py
Choose an option:
1. Embed string in bitmap.
2. Extract string from bitmap.

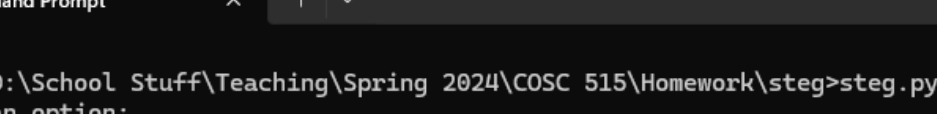
Enter the number corresponding to your desired option: 1

Enter the string to embed: This is a test. This is only a test. Although if I type too much, it's going to get cut off.

Your input is too large for the destination image and will be truncated to the following substring:
This is a test. This is only a test. Although if I type too much, it's going to get

(steg) D:\School Stuff\Teaching\Spring 2024\COSC 515\Homework\steg>|
```

This screenshot shows an example of extracting the string from the image that was created in the previous screenshot.



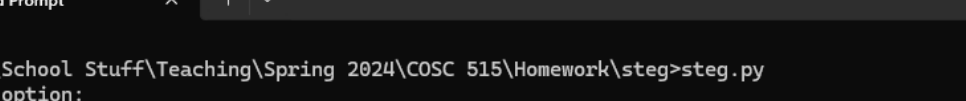
```
Command Prompt
(steg) D:\School Stuff\Teaching\Spring 2024\COSC 515\Homework\steg>steg.py
Choose an option:
1. Embed string in bitmap.
2. Extract string from bitmap.

Enter the number corresponding to your desired option: 2

The following string is embedded in the image:
This is a test. This is only a test. Although if I type too much, it's going to get

(steg) D:\School Stuff\Teaching\Spring 2024\COSC 515\Homework\steg>
```

This screenshot shows an example of trying to embed a string into an image that is too small to store any text. Note that there is no output image generated.



```
Command Prompt
(steg) D:\School Stuff\Teaching\Spring 2024\COSC 515\Homework\steg>steg.py
Choose an option:
1. Embed string in bitmap.
2. Extract string from bitmap.

Enter the number corresponding to your desired option: 1

Enter the string to embed: hi

The destination image does not have enough space to store any text. No output image will be generated.

(steg) D:\School Stuff\Teaching\Spring 2024\COSC 515\Homework\steg>
```

This screenshot shows an example of trying to extract a string from an image that does not have a null terminating character.

```
(steg) D:\School Stuff\Teaching\Spring 2024\COSC 515\Homework\steg>steg.py
Choose an option:
1. Embed string in bitmap.
2. Extract string from bitmap.

Enter the number corresponding to your desired option: 2

WARNING: No null terminator was found when extracting text from the image. This means that steganography was likely not
applied to the input image, and thus, the output string is likely arbitrary nonsense.

The output string created from the image is:
yyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyy
(steg) D:\School Stuff\Teaching\Spring 2024\COSC 515\Homework\steg>
```