

## CS241 – Data Organization Fall 2020

### Programming Assignment #6: Steganography Due by 11:59 PM on Friday, October 16

In this assignment, you will write a more complex C program that includes processing input, using control structures, and bitwise operations.

Encryption is the science of hiding information. Encryption allows Alice and Bob to communicate without their adversary Eve (the eavesdropper) reading and understanding the “secret message”. For example, ROT13 will transform the secret message “helloworld” into “uryybjbeyq”. Although the message is secret, Eve is still able to observe that communication is taking place between Alice and Bob, and this information alone can be very valuable.

Steganography, in contrast to encryption, is the science of preventing others from learning information by hiding that information in “plain sight”. The secret message is not scrambled, or encrypted, but rather it is mixed in with other non-secret information in such a way that, although it is easy to read, people are not even aware that the secret message is there. In this scenario, Eve does not even realize that communication is occurring between Alice and Bob.

Log into the host **linux.unm.edu**. Create a directory for this assignment using the following one-line input to the Unix shell.

```
$ cd cs241; mkdir prog06; chmod 700 prog06; cd prog06
```

The input for your program will be a text file containing a large amount of English. Typically, an English sentence ends with a period (aka, dot). Many years ago, when people used mechanical typewriters, the proper form was to place one space between words in a sentence, but two spaces after the period at the end of the sentence. This rule is no longer taught, since word processors automatically control spacing.

Your program must extract the “secret message” from the input file. The message is hidden inside the file using the following scheme. The message is hidden in binary notation, as a sequence of 0’s and 1’s. Each block of 8-bits is the ASCII encoding of a single character in the “secret message”. Your program will scan the input, and for each period (aka, dot) encountered, your program will count the number of spaces (ASCII 0x20) that immediately follow the dot. If the number of spaces is 0, then your program will ignore that dot completely. If the number of spaces is 1, then that corresponds to a single 0 bit of the secret message. If the number of spaces is 2, then that corresponds to a single 1 bit of the secret message. Finally,

if the number of spaces is 3 or more, then that indicates that there are no more characters in the secret message.

Your program must scan the input file, and output to stdout the plain text secret message. If your program successfully completes its task, then the program should return an exit status of 0 to the operating system. If the number of bits in the message is not a multiple of 8 (8 bits for each ASCII character), then your program should return an exit status of 1. If the input file does not contain the “message over” signal, i.e., a dot followed by 3 or more spaces, then your program should return an exit status of 2.

You can copy the files provided for this assignment using the following command (do not forget to include the dot symbol at the end of this command):

```
$ cp ~soraya/cs241_Soraya/programs/steganography/handouts/* .
```

I have provided several input files from which you should be able to extract the “secret message”. I have also provided input files where the “secret message” is incorrectly encoded, i.e., your program should return a non-zero exit status. You can write your program `foo.c` to read input from stdin (e.g., using `getchar()`), and then test your program on the given files using input redirection, as follows:

```
$ gcc foo.c; ./a.out < goodInputFile
```

Your program should output the “secret message” to stdout.

This program is an example of “stream processing”. You will read the input file, character by character, in a single pass. You **must not use arrays** of any kind, since there is no need to do so, and using arrays is simply inefficient (i.e., overkill). I strongly suggest that before writing a single line of code, not even the include or the main signature, you think of the algorithm you will use to process each character and in particular every dot and spaces after the dot. Understanding the problem clearly and even running a simple example by hand, will save you time at debugging. How are you going to store each bit of each character of the secret message?

You should be sure to include **THE NAME OF THE AUTHOR OF THE PROGRAM** in a comment at the top of your source code file, for this and all other assignments in this course.

Your source code must use proper **style**, that is variables should be well named (name is not too short, not too long, and is meaningful), and bodies of loops, if’s, etc.. should be properly indented. Refer to the coding style file for this class, published on Learn under Coding Standards in a file named: `cs241_codingStandards2020.pdf`.

Create a `.c` file for this assignment and name it using your last name and the initial of your first name, like this: **lastName\_initialFirstName\_secret.c**. Submit this file for grading to Learn in the place of this assignment.