Diary

* Part 1:
  + Cameron Reid:
    - Program design
    - Implementation
    - Testing
  + Kaley Rittichier:
    - Program design
    - Implementation
    - Testing
* Part 2:
  + Cameron Reid:
    - Testing
  + Kaley Rittichier:
    - Program design
    - Implementation
    - Testing
    - Debugging
* Part 3A:
  + Cameron Reid:
    - Testing
  + Kaley Rittichier:
    - Program design
    - Implementation
    - Testing
    - Debugging
* Part 3B:
  + Cameron Reid:
    - Program design
    - Implementation
    - Testing
    - Debugging
  + Kaley Rittichier:
    - Testing
    - Debugging
* Part 3 Analysis:
  + “clear.txt” = 478 bytes
  + “coded.txt” = 2,069 bytes
  + The coded file is over four times the size of the clear file. This is due to the fact that the binary codes is read as characters so for any given character we have the number of that character multiplied by the number of bits in its binary representation. The number of bits per binary representation ranges from 3 to 9 for more frequent to less frequent characters respectively. Because there are more characters with fewer in bits in their binary character representation, we are closer to the 3 than the 9.
  + Example of how this can be done by hand:
    - Frequency of LF = 15
    - Number of bits per binary representation in the code table = 5
    - Therefore, LF contributes to 15\*5 = 75 of the characters in “coded.txt”
    - You can continue this pattern for all the characters and then multiply that number by the number of bits in the basic ASCII set, i.e. 8.
* Part 4A:
  + Cameron Reid:
    - Program design
    - Implementation
    - Testing
    - Debugging
  + Kaley Rittichier:
    - Testing
* Part 4B:
  + Cameron Reid:
    - Program design
    - Implementation
    - Testing
    - Debugging
  + Kaley Rittichier:
    - Testing
    - Debugging
* Part 4 Analysis:
  + < compare the size of the files clear.txt and codedalt.txt. >
  + clear.txt: 463 bytes (part 4 analysis was done on a Linux operating system, which involved some file manipulation that accounts for the difference in file size)
  + coded\_binary.txt: 259 bytes
  + Although part 4 encodes basically the same information as part 3, the encoded file is now only ~55% the size of the original file because the information is encoded as bits rather than ASCII “1” and “0”, which require 8 bits of storage. For example:
    - the string “ae” requires 16 bits to store, 8 for each character;
    - Using the ASCII encoding, it would require 32 bits for “a” (encoded “1110”) and 24 bits for “e” (encoded “001”)
    - Storing those encodings with bits allows us to store the string with only 7 bits: less than a single character of ASCII.