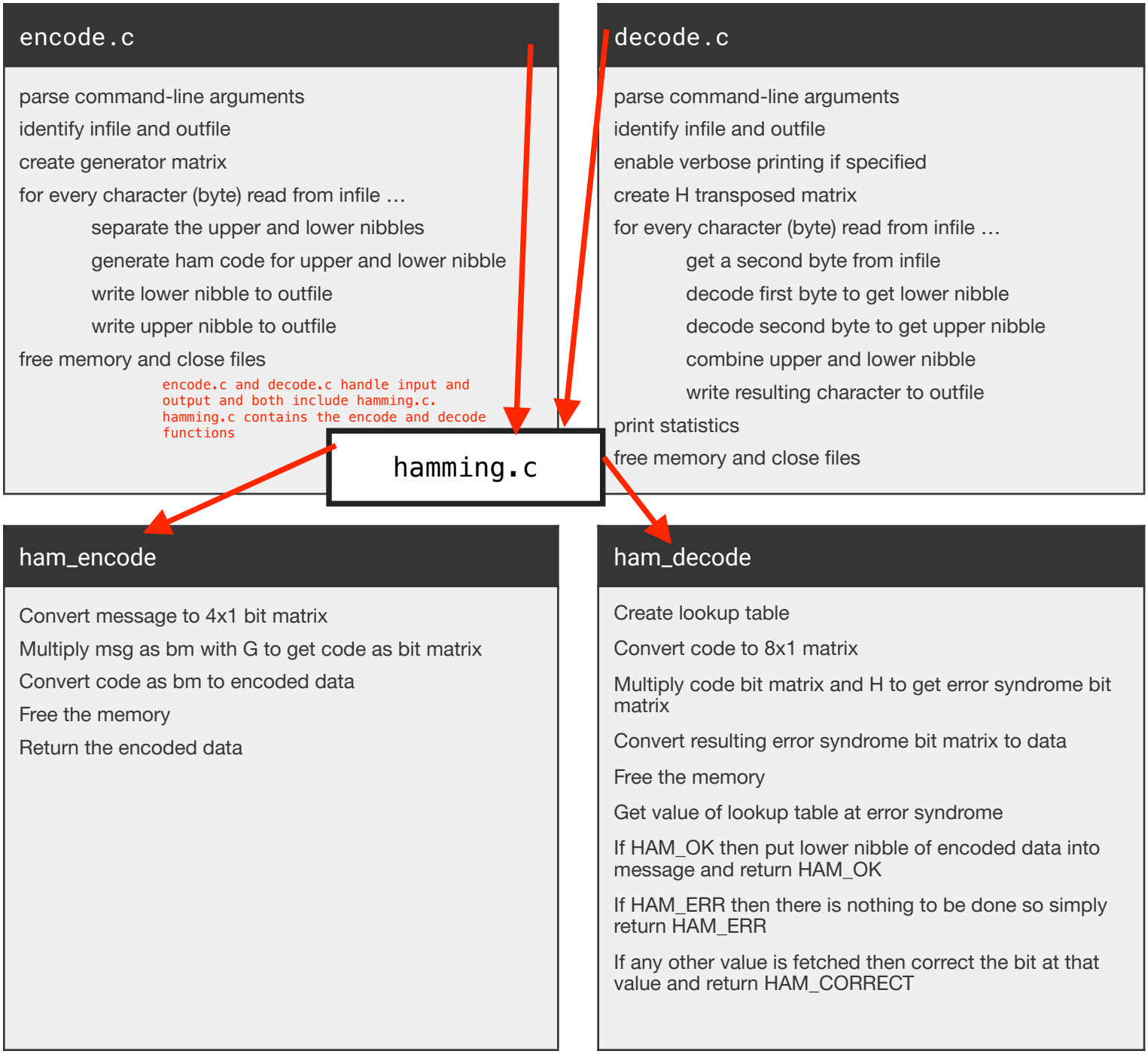
Assignment 5

Design Document

Purpose

Create a program that will encode a given message using Hamming(8, 4) codes into encoded data, and create a corresponding program to decode the encoded data back into a message.

Layout/Structure and Pseudocode for a Clear Description/Explanation of How Each Part of the Program Should Work:



MORE PSEUDOCODE I DRAFTED THROUGHOUT MY DESIGN PROCESS

encode.c

```
infile is stdin by default
outfile is stdout by default
for every option given ...

  if option is ‘h’ then print help message, close files, and return
  if option is’ i’ then update the infile to file specified by optarg
  if option is ‘o’ then update the outfile to file specified by optarg
  if option is unknown then print help message, close files, and return


initialize the 4x8 generator matrix (with all bit values equal to zero)
flip the correct bits from 0 to 1 such that generator matrix looks like so:
1 0 0 0 0 1 1 1
0 1 0 0 1 0 1 1
0 0 1 0 1 1 0 1
0 0 0 1 1 1 1 0


for every character (byte) read from the infile

  get the lower nibble of the byte (use function provided in asgn PDF)
  encode the lower nibble using ham_encode
  write the encoded lower nibble to the outfile

  get the upper nibble of the byte (use function provided in asgn PDF)
  encode the upper nibble using ham_encode
  write the encoded lower nibble to the outfile

free memory
close files
return
```

```
decode.c

main

    parse command-line arguments

    create Ht

    for every character read from the infile ...

         immediately read a second character (to get the corresponding code for the upper nibble)

         switch (decode the first character to get lower nibble) {
         case HAM_OK: do nothing
         case HAM_ERR: increment counter for uncorrected errors
         case HAM_CORRECT: increment counter for corrected errors

         switch (decode the second character to get upper nibble) {
         case HAM_OK: do nothing
         case HAM_ERR: increment counter for uncorrected errors
         case HAM_CORRECT: increment counter for corrected errors

         pack the lower and upper nibble back into one byte to get the original decoded character

         write the decoded character (byte) back to the outfile

         increment the counter for total processed bytes by TWO because two bytes have been processed

    now that the file has been decoded ...
    if verbose printing is enabled then print statistics

    delete memory and close files

    return
```

ham_decode

```
HAM_STATUS ham_decode(BitMatrix *Ht, uint8_t code, uint8_t *msg)

    Convert code to 8x1 matrix

    Multiply code bit matrix and H to get error syndrome bit matrix

    Convert resulting error syndrome bit matrix to data

    Create lookup table

    switch (get value of lookup table at error syndrome)

         case HAM_OK then put lower nibble of encoded data into message and return HAM_OK

         case HAM_ERR then there is nothing to be done so simply return HAM_ERR

         case any other value is fetched then correct the bit at that value and return HAM_CORRECT

    Free the memory

  return
```