Pre-Lab Questions

1) Calculate Hamming Codes for $0000_2 - 1111_2$ using the generator matrix.
Given a message, $\vec{m}$, 4-bits, $\vec{c} = \vec{m}G$ where $\vec{c}$ is the hamming code &

$$G = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 1 & 1 & 1 \\ 0 & 1 & 0 & 0 & 1 & 0 & 1 & 1 \\ 0 & 0 & 1 & 0 & 1 & 1 & 0 & 1 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 & 0 \end{bmatrix}$$

a) $\vec{c} = [0\,0\,0\,0]\,G \pmod 2$
$= [0\,0\,0\,0] \pmod 2$
$= [0000]$
$= 0000_2$

b) $\vec{c} = [0001]\,G \pmod 2$
$= [00011110] \pmod 2$
$= [00011110]$
$= 01111000_2$

c) $\vec{c} = [0010]\,G \pmod 2$
$= [00101101] \pmod 2$
$= [00101101]$
$= 10110100_2$

d) $\vec{c} = [0011]\,G \pmod 2$
$= [0011\ 22\ 11] \pmod 2$
$= [0011\ 00\ 11]$
$= 1100\,1100_2$

e) $\vec{c} = [0100]\,G \pmod 2$
$= [010010\,11] \pmod 2$
$= [010010\,11]$
$= 11010010_2$

f) $\vec{c} = [0101]\,G \pmod 2$
$= [01012121] \pmod 2$
$= [01010101]$
$= 10101010_2$

g) $\vec{c} = [0110]\,G \pmod 2$
$= [0110\,2112] \pmod 2$
$= [01100110]$
$= 01100110_2$

h) $\vec{c} = [0111]\,G \pmod 2$
$= [0111\,3222] \pmod 2$
$= [01111\,000]$
$= 00011110_2$

i) $\vec{c} = [1000]\,G \pmod 2$
$= [10000111] \pmod 2$
$= [10000\,111]$
$= 11100001_2$

j) $\vec{c} = [1001]\,G \pmod 2$
$\vec{c} = [1\,0\,0\,1\,1\,2\,2\,1] \pmod 2$
$= [10011001]$
$= 10011001_2$

k) $\vec{c} = [1010]\,G \pmod 2$
$= [10101212] \pmod 2$
$= [10101010]$
$= 01010101_2$

l) $\vec{c} = [1011]\,G \pmod 2$
$= [1011\,2322] \pmod 2$
$= [1011\,0100]$
$= 00101101_2$

m) $\vec{c} = [1100]G \pmod 2$
$= [11001122] \pmod 2$
$= [11001100]$
$= 00110011_2$

n) $\vec{c} = [1101]G \pmod 2$
$= [1101\ 22\ 32] \pmod 2$
$= [1101\ 00\ 10]$
$= 01001011_2$

o) $\vec{c} = [1110]G \pmod 2$
$= [1110\ 22\ 23] \pmod 2$
$= [1110\ 0001]$
$= 10000111_2$

p) $\vec{c} = [1111]G \pmod 2$
$= [1111\ 33\ 33] \pmod 2$
$= [1111\ 1111]$
$= 1111\ 1111_2$

2) Decode the following codes.

a) 11100011_2

$$H = \begin{bmatrix} 0 & 1 & 1 & 1 & 1 & 0 & 0 & 0 \\ 1 & 0 & 1 & 1 & 0 & 1 & 0 & 0 \\ 1 & 1 & 0 & 1 & 0 & 0 & 1 & 0 \\ 1 & 1 & 1 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}$$

$\vec{e} = \vec{c}H^T$
$= [11\ 000\ 111]H^T \pmod 2$
$= [1\ 2\ 3\ 3] \pmod 2$
$= [1\ 0\ 1\ 1]$

$\vec{e}$ matches the second row of H. Thus, the code has an error at the second element in $\vec{c}$. To correct this, we simply flip the value of the second element, giving $\vec{c} = [10000111]$, giving us $\vec{m} = [1000]$.

b) 11011000_2

$\vec{e} = \vec{c}H^T$
$= [00011011]H^T \pmod 2$
$= [2121] \pmod 2$
$= [0101]$

We cannot correct the error since more than 1 bit has been flipped. $\vec{e}$ does not match any of the H-columns.

3) Complete the look-up table.

[0000]	0	HAM_OK
[1000]	1	~~5~~ 4
[0100]	2	~~6~~ 5
[1100]	3	HAM_ERR
[0010]	4	~~7~~ 6
[1010]	5	HAM_ERR
[0110]	6	HAM_ERR
[1110]	7	~~4~~ 3
[0001]	8	~~8~~ 7
[1001]	9	HAM_ERR
[0101]	10	HAM_ERR
[1101]	11	~~3~~ 2
[0011]	12	HAM_ERR
[1011]	13	~~2~~ 1
[0111]	14	~~1~~ 0
[1111]	15	HAM_ERR

Michael Nguyen
mnguy181@ucsc.edu
2/04/2021

CSE13S Winter 2021
Assignment 4: Hamming Codes
Design Document

PURPOSE

The purpose of this assignment will be to implement a Hamming code library that uses the Hamming (8,4) code as explained in the lab document. In essence, I will be creating two small programs for this assignment, one of which will be used for generating Hamming codes while the other part will be used for decoding Hamming codes. To implement matrices G and H as described in the lab document, I will be utilizing a bit matrix ADT. Both small programs will read from stdin (by default) or a file and will write to stdout (by default) or another output file specified. Therefore, like the last assignment, my main() will support command-line arguments, -i and -o, in order to specify the input and output files.

According to the lab document, source code for error.c (which will be responsible for injecting noise into my Hamming codes) will be provided at a rate specified by the command line argument -e rate (default as 0.01). The seed will be specified with -s seed (must be positive).

TOP LEVEL FOR THE HAMMING CODES GENERATOR

The goals of the hamming codes generator program will be to parse the command-line options with the getopt() command and open any input and/or output files. I will then initialize the hamming code module with ham_init(), read a byte from the specified input file or stdin, generate the codes until all data has been read from the file, and then free all memory allocated by the Hamming Code module. The program will then end by closing both input and output files.

The top level design of my code for generator.c is given by the following pseudocode.

```
Define the command line options
main()
    Initialize and set the default values of infile, outfile
    While loop for getopt(), opening any input or output files
    Error handle for invalid input files
    Initialize the hamming code module with ham_init()
    While loop() ending when fgetc() is -1
        fgetc() each byte from stdin or input file
        ham_gen() to generate Hamming codes
        fputc() lower nibble
        fputc() upper nibble
    ham_destroy() to free all allocated memory
    Close files
```

TOP LEVEL FOR THE HAMMING CODES DECODER

The goals of the hamming codes decoder program will be to parse the command-line options with the getopt() command and open any input and/or output files. I will then initialize the hamming code module with ham_init(), read 2 bytes from the specified input file or stdin, output the reconstructed byte to the output file, decode each byte pair until all data has been read from the file, and then free all memory allocated by the Hamming Code module. This program will also print some statistics to stderr which include the following: The number of total bytes processed, the number of uncorrected errors, the number of corrected errors, and the error rate. The program will then end by closing both input and output files.

The top level design of my code for decoder.c is given by the following pseudocode.

```
Define the command line options -i and -o
main()
        Initialize and set the default values of infile, outfile
        While loop() for getopt(), opening any input or output files
        Error handle for invalid input files
        Initialize using ham_init() for hamming code module
        While Loop() ending when fgetc() is -1
                fgetc() two times to read two bytes from infile or stdin
                First fgetc() is lower nibble, second is upper nibble, store in variables
                ham_dec() lower nibble and upper nibble
                Increment a variable count to keep track of number of bytes processed
                Keep count of hamming codes that required correction
                Keep count of hamming codes that could not be corrected
                fputc() reconstructed byte to output file or stdout
        ham_destroy() to free all memory
        fprintf() to stderr total bytes processed
        fprintf() to stderr uncorrected errors number
        fprintf() to stderr number of corrected errors
        fprintf() to stderr error rate = uncorrected errors/total number of bytes processed
        Close in and output file
```

PSEUDOCODE for bm.c which will contain my implementation of the bit matrix ADT will be provided below. Please note that bm.h, the header file, will be provided (and I will not be modifying this file).

```
Struct BitMatrix
        Declare the fields rows, cols, **mat
bm_create()
        Allocate memory for the BitMatrix using calloc
        Initialize and set m->rows, m->cols
        Allocate the memory for the matrix using calloc()
        For loop() that allocates memory for each column in the BitMatrix
bm_delete()
        For loop() iterating through each row, freeing each column in BitMatrix
        Free the matrix
        Free the BitMatrix pointer, setting pointer to NULL
bm_rows()
```

Returns m->rows

bm_cols()

Returns m->cols

bm_set_bit()

Initialize mask to 1 << columns % 8

mat[row][col/8] = mat[row][col/8] OR mask

bm_clr_bit()

Initialize mask to !(1 << columns % 8)

mat[row][col/8] = mat[row][col/8] AND mask

bm_get_bit()

Initialize mask to 1 << columns % 8

Set mat[row][col/8] to its value AND mask

Return the bit right shifted by (columns % 8)

bm_print()

For loop that iterates through each matrix's row

Print each column

PSEUDOCODE for hamming.c which will contain my implementation of the Hamming Code module will be provided below. Please note that hamming.h, the header file, will be provided (and I will not be modifying this file).

Typedef enum ham_rc

Define ham_err = -1, ham_ok = 0, ham_err_ok = 1

Global variable bitmatrix g

Global variable bitmatrix ht

ham_rc ham_init()

bm_create() g and ht

set_bit() g at mat[0][0]

For loop() to initialize g bit matrix adt

For loop() to initialize h bit matrix adt

If -1:

Return ham_err

Else:

Return ham_ok

ham _rc ham_destroy()

For loop() to free bitmatrix g

For loop() to free bitmatrix ht

Free pointers

Ham_rc ham_gen()

Ham_rc ham_dec()