Pre-lab questions

1.
    (0, 0 , 0, 0) x G matrix = (0, 0, 0, 0, 0, 0, 0, 0)
    (0, 0, 0, 0, 0, 0, 0, 0) % 2 = (0, 0, 0, 0, 0, 0, 0, 0)

    (0, 0 , 0, 1)
    Flipped x G matrix = (1, 0, 0, 0, 0, 1, 1, 1)
    (1, 0, 0, 0, 0, 1, 1, 1) % 2 = (1, 0, 0, 0, 0, 1, 1, 1)
    Flipped back = (1 1 1 0 0 0 0 1)

    (0, 0 , 1, 0)
    flipped x G matrix = (0, 1, 0, 0, 1, 0, 1, 1)
    (0, 1, 0, 0, 1, 0, 1, 1) % 2 = (0, 1, 0, 0, 1, 0, 1, 1)
    Flipped back = (1 1 0 1 0 0 1 0)

    (0, 0 , 1, 1)
    Flipped x G matrix = (1, 1, 0, 0, 1, 2, 2, 2)
    (1, 1, 0, 0, 1, 2, 2, 2) % 2 = (1, 1, 0, 0, 1, 0, 0, 0)
    Flipped back = (0 0 0 1 0 0 1 1)

    (0, 1 , 0, 0)
    Flipped x G matrix = (0, 0, 1, 0, 1, 1, 0, 1)
    (0, 0, 1, 0, 1, 1, 0, 1) % 2 = (0, 0, 1, 0, 1, 1, 0, 1)
    Flipped back = (1 0 1 1 0 1 0 0)

    (0, 1 , 0, 1)
    Flipped x G matrix = (1, 0, 1, 0, 1, 2, 1, 2)
    (1, 0, 1, 0, 1, 2, 1, 2) % 2 = (1, 0, 1, 0, 1, 0, 1, 0)
    Flipped back = (0 1 0 1 0 1 0 1)

    (0, 1, 1, 0) x G matrix = (0, 1, 1, 0, 2, 1, 1, 2)
    (0, 1, 1, 0, 2, 1, 1, 2) % 2 = (0, 1, 1, 0, 0, 1, 1, 0)

    (0, 1 , 1, 1)
    Flipped x G matrix = (1, 1, 1, 0, 2, 2, 2, 3)
    (1, 1, 1, 0, 2, 2, 2, 3) % 2 = (1, 1, 1, 0, 0, 0, 0, 1)
    Flipped back = (1 0 0 0 0 1 1 1)

    (1, 0 , 0, 0)
    Flipped x G matrix = (1, 0, 0, 0, 0, 1, 1, 1)
    (1, 0, 0, 0, 0, 1, 1, 1) % 2 = (1, 0, 0, 0, 0, 1, 1, 1)
    Flipped back = (1 1 1 0 0 0 0 1)

(1, 0 , 0, 1) x G matrix = (0, 1, 0, 1, 2, 1, 2, 1)
(0, 1, 0, 1, 2, 1, 2, 1) % 2 =(0, 1, 0, 1, 0, 1, 0, 1)

(1, 0 , 1, 0) x G matrix = Flipped x G matrix = (0, 1, 0, 1, 2, 1, 2, 1)
(0, 1, 0, 1, 2, 1, 2, 1) % 2 =(0, 1, 0, 1, 0, 1, 0, 1)
Flipped back = (1 0 1 0 1 0 1 0)

(1, 0 , 1, 1)
Flipped x G matrix = (1, 1, 0, 1, 2, 2, 3, 2)
(1, 1, 0, 1, 2, 2, 3, 2) % 2 = (1, 1, 0, 1, 0, 0, 1, 0)
Flipped back = (0 1 0 0 1 0 1 1)

(1, 1 , 0, 0)
Flipped x G matrix = (0, 0, 1, 1, 2, 2, 1, 1)
(0, 0, 1, 1, 2, 2, 1, 1) % 2 = (0, 0, 1, 1, 0, 0, 1, 1)
Flipped back = (1 1 0 0 1 1 0 0)

(1, 1 , 0, 1)
Flipped x G matrix = (1, 0, 1, 1, 2, 3, 2, 2)
(1, 0, 1, 1, 2, 3, 2, 2) % 2 = (1, 0, 1, 1, 0, 1, 0, 0)
Flipped back = (0 0 1 0 1 1 0 1)

(1, 1 , 1, 0)
Flipped x G matrix = (0, 1, 1, 1, 3, 2, 2, 2)
(0, 1, 1, 1, 3, 2, 2, 2) % 2 = (0, 1, 1, 1, 1, 0, 0, 0)
Flipped back = (0 0 0 1 1 1 1 0)

(1, 1 , 1, 1) x G matrix = (1, 1, 1, 1, 3, 3, 3, 3)
(1, 1, 1, 1, 3, 3, 3, 3) % 2 = (1, 1, 1, 1, 1, 1, 1, 1)

2. (1,1,1,0, 0, 0, 1, 1) x transpose of H matrix = (2, 2, 3, 4)
   (2, 2, 3, 4) % 2 = (0, 0, 1, 0)
   7th element was flipped
   Flipping it back and x transpose of H matrix = (2, 2, 2, 4)
   (2, 2, 2, 4) % 2 = 0 (no errors)

   (1,1,0, 1, 1, 0, 0, 0) x transpose of H matrix = (3, 2, 3, 2)
   (3, 2, 3, 2) % 2 = (1, 0, 1, 0)
   Cannot be fixed as (1, 0, 1, 0) is not in the transpose matrix H

3.

0	0
1	4
2	5
3	HAM_ERR
4	6
5	HAM_ERR
6	HAM_ERR
7	3
8	7
9	HAM_ERR
10	HAM_ERR
11	2
12	5
13	1
14	0
15	HAM_ERR

Ham_rc ham_init(void)

Creates and initializes the G and H matrices.
Allocating memory for pointers
G and H should be static variables

$$\boldsymbol{G} = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 1 & 1 & 1 \\ 0 & 1 & 0 & 0 & 1 & 0 & 1 & 1 \\ 0 & 0 & 1 & 0 & 1 & 1 & 0 & 1 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 & 0 \end{pmatrix}. \qquad \boldsymbol{H} = \begin{pmatrix} 0 & 1 & 1 & 1 & 1 & 0 & 0 & 0 \\ 1 & 0 & 1 & 1 & 0 & 1 & 0 & 0 \\ 1 & 1 & 0 & 1 & 0 & 0 & 1 & 0 \\ 1 & 1 & 1 & 0 & 0 & 0 & 0 & 1 \end{pmatrix}.$$

Return ham_err if fails
Return ham_ok if all good

G is used for encoding and H is used for decoding
Want to use transpose for H when decoding

void ham_rc ham_destroy(void)

Free G and H

Frees memory that you allocated

ham_rc ham_encode(uint8_t data, uint8_t *code)

Access the nibble of data

Multiply our 4 bits by the G matrix

Mod 2 the result

Update *code with the result

Return HAM_OK if successful

Otherwise HAM_ERR

ham_rc ham_decode(uint8_t code, uint8_t *data)

Multiply hamming code with the transpose of the H matrix

Mod 2 the result

Match the result to the look-up table

Correct the data by the index in look-up table

Return HAM_ERR_OK if correctable

If no correction return HAM_OK

Otherwise HAM_ERR

Bit Matrix

BitMat *bm_create(uint32_t rows, uint32_t cols)

Making space in memory for the matrix

Matrix space is rows x columns

Returns a pointer to a bitmap

void bm_delete(BitMat **m)

Frees up the memory allocated for bitmap and ADT

Frees the matrix itself

Set pointer to NULL after freeing

uint32_t bm_rows(BitMat *m)

Returns bit rows from struct

uint32_t bm_cols(BitMat *m)

Returns bit cols from struct

void bm_set_bit(BitMat *m, uint32_t row, uint32_t col)

Access the bit and then set it

Setting a bit you can mask the bit position with a 1 (everything else is 0)

And OR them together
Setting a bit makes it = 1

void bm_clr_bit(BitMat *m, uint32_t row, uint32_t col)

Access a bit and then clear it
Clearing a bit you can mask the bit position with a 1 (everything else is 0)
Invert it (~ in C)
Then AND the mask and byte together
Result = byte & mask
Clearing a bit makes it = 0

uint8_t bm_get_bit(BitMat *m, uint32_t row, uint32_t col)

1 if set and 0 if not set
Mask is going to put a 1 into the position of the bit we want
Left shift over to bit position
Result = byte & mask
Shift left back same amount of original position
To get bit into leftmost bit

void bm_print(BitMat *m)

Loop through the matrix and print it out, similar to last assignment

Generator Program

Go through command line options and open input/output files
Initialize hamming code

Loop through each byte from file:

- Read a byte from the input file with fgetc()
- Generate hamming codes for upper and lower nibble with ham_encode()
- Use helper functions to get lower and upper nibble
- Output to stdout
- End loop when all data has been read from input file

Free the memory allocated using ham_destroy()
Close the input and output files using fclose()

Decoder Program (similar to generator)

Go through command line options and open input/output files
Initialize hamming code

Loop through each pair of bytes in file:

Read two bytes from input file (first is lower nibble, second is upper nibble)
Decode with ham_decode()
Reconstruct the original byte
Write the reconstructed byte with fputc()
End loop when when all data has been read from input (EOF)

Use ham_destry to free memory allocated

Print stats to stderr:

Total bytes processed: use a counter to keep track by bytes read
Uncorrected errors: number of times HAM_ERR was returned
Corrected errors: number of times HAM_ERR_OK was returned
Error rate: (number of uncorrected errors) / (total number of bytes)

Close input and output files