Pre-lab Part 1

⚠ Pre-labs are part of the design document, and answers to the questions should be at the head of the document.

1. Calculate Hamming codes for 0000_2–1111_2 using the generator matrix. Show your work. *Hint*: Convert your codes to hex.

Need to encode:

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

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

a) encoding 0000

$\underline{A}=[0000]$

$\underset{1\times4}{\underline{A}}\cdot\underset{4\times8}{\underline{H}}=\underset{1\times8}{\vec{C}}$

$$[0000]\cdot\begin{pmatrix}1&0&0&0&0&1&1&1\\0&1&0&0&1&0&1&1\\0&0&1&0&1&1&0&1\\0&0&0&1&1&1&1&0\end{pmatrix}=(C_{00},C_{01},C_{02},C_{03},C_{04},C_{05},C_{06},C_{07})$$

$C_{00}=0\cdot1+0\cdot0+0\cdot0+0\cdot0=0$
$C_{01}=0\cdot0+0\cdot1+0\cdot0+0\cdot0=0$
$C_{02}=0\cdot0+0\cdot0+0\cdot1+0\cdot0=0$
$C_{03}=0\cdot0+0\cdot0+0\cdot0+0\cdot1=0$
$C_{04}=0\cdot0+0\cdot1+0\cdot1+0\cdot1=0$
$C_{05}=0\cdot1+0\cdot0+0\cdot1+0\cdot1=0$
$C_{06}=0\cdot1+0\cdot1+0\cdot0+0\cdot1=0$
$C_{07}=0\cdot1+0\cdot1+0\cdot1+0\cdot0=0$

$\rightarrow \boxed{\vec{c}=[0\,0\,0\,0\,0\,0\,0\,0]}$

b) encoding MSB 0001_2 LSB $\rightarrow 1_{10}$

LSB MSB
$\underline{A}=[1000]$

$\underset{1\times4}{\underline{A}}\cdot\underset{4\times8}{\underline{H}}=\underset{1\times8}{\vec{C}}$

$$[1000]\cdot\begin{pmatrix}1&0&0&0&0&1&1&1\\0&1&0&0&1&0&1&1\\0&0&1&0&1&1&0&1\\0&0&0&1&1&1&1&0\end{pmatrix}=(C_{00},C_{01},C_{02},C_{03},C_{04},C_{05},C_{06},C_{07})$$

C_{00}	C_{01}	C_{02}	C_{03}	C_{04}	C_{05}	C_{06}	C_{07}
1	0	0	0	0	1	1	1

⇓

LSB MSB
$\boxed{\vec{c}=[1\,0\,0\,0\,0\,1\,1\,1]}$

c) encoding MSB 0010_2 LSB $\rightarrow 2_{10}$

LSB MSB
$\underline{A}=[0100]$

$\underset{1\times4}{\underline{A}}\cdot\underset{4\times8}{\underline{H}}=\underset{1\times8}{\vec{C}}$

$$[0100]\cdot\begin{pmatrix}1&0&0&0&0&1&1&1\\0&1&0&0&1&0&1&1\\0&0&1&0&1&1&0&1\\0&0&0&1&1&1&1&0\end{pmatrix}=(C_{00},C_{01},C_{02},C_{03},C_{04},C_{05},C_{06},C_{07})$$

C_{00}	C_{01}	C_{02}	C_{03}	C_{04}	C_{05}	C_{06}	C_{07}
0	1	0	0	1	0	1	1

⇓

LSB MSB
$\boxed{\vec{c}=[0\,1\,0\,0\,1\,0\,1\,1]}$

d) encoding MSB 0011_2 LSB $\rightarrow 3_{10}$

LSB MSB
$\underline{A}=[1100]$

$\underset{1\times4}{\underline{A}}\cdot\underset{4\times8}{\underline{H}}=\underset{1\times8}{\vec{C}}$

$$[1100]\cdot\begin{pmatrix}1&0&0&0&0&1&1&1\\0&1&0&0&1&0&1&1\\0&0&1&0&1&1&0&1\\0&0&0&1&1&1&1&0\end{pmatrix}=(C_{00},C_{01},C_{02},C_{03},C_{04},C_{05},C_{06},C_{07})$$

C_{0x}	C_{00}	C_{01}	C_{02}	C_{03}	C_{04}	C_{05}	C_{06}	C_{07}
$A\cdot H_x$	0	1	1	0	1	1	2	2
C_{0x} MOD 2	0	1	1	0	1	1	0	0

⇓

LSB MSB
$\boxed{\vec{c}=[0\,1\,1\,0\,1\,1\,0\,0]}$

e) encoding MSB 0100_2 LSB $\rightarrow 4_{10}$

LSB MSB
$\underline{A}=[0010]$

$\underset{1\times4}{\underline{A}}\cdot\underset{4\times8}{\underline{H}}=\underset{1\times8}{\vec{C}}$

$$[0010]\cdot\begin{pmatrix}1&0&0&0&0&1&1&1\\0&1&0&0&1&0&1&1\\0&0&1&0&1&1&0&1\\0&0&0&1&1&1&1&0\end{pmatrix}=(C_{01},C_{02},C_{03},C_{04},C_{05},C_{06},C_{07},C_{08})$$

C_{0x}	C_{01}	C_{02}	C_{03}	C_{04}	C_{05}	C_{06}	C_{07}	C_{08}
$A\cdot H_x$	0	0	1	0	1	1	0	1

⇓

LSB MSB
$\boxed{\vec{c}=[0\,0\,1\,0\,1\,1\,0\,1]}$

f) encoding MSB 0101_2 LSB $\rightarrow 5_{10}$

LSB MSB
$\underline{A}=[1010]$

$\underset{1\times4}{\underline{A}}\cdot\underset{4\times8}{\underline{H}}=\underset{1\times8}{\vec{C}}$

$$[1010]\cdot\begin{pmatrix}1&0&0&0&0&1&1&1\\0&1&0&0&1&0&1&1\\0&0&1&0&1&1&0&1\\0&0&0&1&1&1&1&0\end{pmatrix}=(C_{01},C_{02},C_{03},C_{04},C_{05},C_{06},C_{07},C_{08})$$

C_{0x}	C_{01}	C_{02}	C_{03}	C_{04}	C_{05}	C_{06}	C_{07}	C_{08}
$A\cdot H_x$	1	0	1	0	1	2	1	2
C_{0x} MOD 2	1	0	1	0	1	0	1	0

⇓

LSB MSB
$\boxed{\vec{c}=[1\,0\,1\,0\,1\,0\,1\,0]}$

g) encoding MSB 0110_2 LSB $\rightarrow 6_{10}$

LSB MSB
$\underline{A}=[0110]$

$\underset{1\times4}{\underline{A}}\cdot\underset{4\times8}{\underline{H}}=\underset{1\times8}{\vec{C}}$

$$[0110]\cdot\begin{pmatrix}1&0&0&0&0&1&1&1\\0&1&0&0&1&0&1&1\\0&0&1&0&1&1&0&1\\0&0&0&1&1&1&1&0\end{pmatrix}=(C_{01},C_{02},C_{03},C_{04},C_{05},C_{06},C_{07},C_{08})$$

C_{0x}	C_{01}	C_{02}	C_{03}	C_{04}	C_{05}	C_{06}	C_{07}	C_{08}
$A\cdot H_x$	0	1	1	0	2	1	1	2
C_{0x} MOD 2	0	1	1	0	0	1	1	0

⇓

LSB MSB
$\boxed{\vec{c}=[0\,1\,1\,0\,0\,1\,1\,0]}$

h) encoding MSB 0111_2 LSB $\rightarrow 7_{10}$

LSB MSB
$\underline{A}=[1110]$

$\underset{1\times4}{\underline{A}}\cdot\underset{4\times8}{\underline{H}}=\underset{1\times8}{\vec{C}}$

$$[1110]\cdot\begin{pmatrix}1&0&0&0&0&1&1&1\\0&1&0&0&1&0&1&1\\0&0&1&0&1&1&0&1\\0&0&0&1&1&1&1&0\end{pmatrix}=(C_{01},C_{02},C_{03},C_{04},C_{05},C_{06},C_{07},C_{08})$$

C_{0x}	C_{01}	C_{02}	C_{03}	C_{04}	C_{05}	C_{06}	C_{07}	C_{08}
$A\cdot H_x$	0	1	1	0	2	1	1	2
C_{0x} MOD 2	0	1	1	0	0	1	1	0

⇓

LSB MSB
$\boxed{\vec{c}=[0\,1\,1\,0\,0\,1\,1\,0]}$

h) encoding MSB 1000_2 LSB $\rightarrow 8_{10}$

LSB MSB
$\underline{A}=[0001]$

$\underset{1\times4}{\underline{A}}\cdot\underset{4\times8}{\underline{H}}=\underset{1\times8}{\vec{C}}$

$$[0001]\cdot\begin{pmatrix}1&0&0&0&0&1&1&1\\0&1&0&0&1&0&1&1\\0&0&1&0&1&1&0&1\\0&0&0&1&1&1&1&0\end{pmatrix}=(C_{01},C_{02},C_{03},C_{04},C_{05},C_{06},C_{07},C_{08})$$

C_{0x}	C_{01}	C_{02}	C_{03}	C_{04}	C_{05}	C_{06}	C_{07}	C_{08}
$A\cdot H_x$	0	0	0	1	1	1	1	0
C_{0x} MOD 2	0	0	0	1	1	1	1	0

⇓

LSB MSB
$\boxed{\vec{c}=[0\,0\,0\,1\,1\,1\,1\,0]}$

I) encoding MSB 1001_2 LSB $\rightarrow 9_{10}$

LSB MSB
$\underline{A}=[1001]$

$\underset{1\times4}{\underline{A}}\cdot\underset{4\times8}{\underline{H}}=\underset{1\times8}{\vec{C}}$

$$[1001]\cdot\begin{pmatrix}1&0&0&0&0&1&1&1\\0&1&0&0&1&0&1&1\\0&0&1&0&1&1&0&1\\0&0&0&1&1&1&1&0\end{pmatrix}=(C_{01},C_{02},C_{03},C_{04},C_{05},C_{06},C_{07},C_{08})$$

C_{0x}	C_{01}	C_{02}	C_{03}	C_{04}	C_{05}	C_{06}	C_{07}	C_{08}
$A\cdot H_x$	1	0	0	1	1	2	2	1
C_{0x} MOD 2	1	0	0	1	1	0	0	1

⇓

LSB MSB
$\boxed{\vec{c}=[1\,0\,0\,1\,1\,0\,0\,1]}$

J) encoding MSB 1010_2 LSB $\rightarrow 10_{10}$

LSB MSB
$\underline{A}=[0101]$

$\underset{1\times4}{\underline{A}}\cdot\underset{4\times8}{\underline{H}}=\underset{1\times8}{\vec{C}}$

$$[0101]\cdot\begin{pmatrix}1&0&0&0&0&1&1&1\\0&1&0&0&1&0&1&1\\0&0&1&0&1&1&0&1\\0&0&0&1&1&1&1&0\end{pmatrix}=(C_{01},C_{02},C_{03},C_{04},C_{05},C_{06},C_{07},C_{08})$$

C_{0x}	C_{01}	C_{02}	C_{03}	C_{04}	C_{05}	C_{06}	C_{07}	C_{08}
$A\cdot H_x$	0	1	0	1	2	1	2	1
C_{0x} MOD 2	0	1	0	1	0	1	0	1

⇓

LSB MSB
$\boxed{\vec{c}=[0\,1\,0\,1\,0\,1\,0\,1]}$

K) encoding MSB 1011_2 LSB $\rightarrow 11_{10}$

LSB MSB
$\underline{A}=[1101]$

$\underset{1\times4}{\underline{A}}\cdot\underset{4\times8}{\underline{H}}=\underset{1\times8}{\vec{C}}$

$$[1101]\cdot\begin{pmatrix}1&0&0&0&0&1&1&1\\0&1&0&0&1&0&1&1\\0&0&1&0&1&1&0&1\\0&0&0&1&1&1&1&0\end{pmatrix}=(C_{01},C_{02},C_{03},C_{04},C_{05},C_{06},C_{07},C_{08})$$

C_{0x}	C_{01}	C_{02}	C_{03}	C_{04}	C_{05}	C_{06}	C_{07}	C_{08}
$A\cdot H_x$	1	1	0	1	2	2	3	2
C_{0x} MOD 2	1	1	0	1	0	0	1	0

⇓

LSB MSB
$\boxed{\vec{c}=[1\,1\,0\,1\,0\,0\,1\,0]}$

L) encoding MSB 1100_2 LSB $\rightarrow 12_{10}$

LSB MSB
$\underline{A}=[0011]$

$\underset{1\times4}{\underline{A}}\cdot\underset{4\times8}{\underline{H}}=\underset{1\times8}{\vec{C}}$

$$[0011]\cdot\begin{pmatrix}1&0&0&0&0&1&1&1\\0&1&0&0&1&0&1&1\\0&0&1&0&1&1&0&1\\0&0&0&1&1&1&1&0\end{pmatrix}=(C_{01},C_{02},C_{03},C_{04},C_{05},C_{06},C_{07},C_{08})$$

C_{0x}	C_{01}	C_{02}	C_{03}	C_{04}	C_{05}	C_{06}	C_{07}	C_{08}
$A\cdot H_x$	0	0	1	1	2	2	1	1
C_{0x} MOD 2	0	0	1	1	0	0	1	1

⇓

LSB MSB
$\boxed{\vec{c}=[0\,0\,1\,1\,0\,0\,1\,1]}$

L) encoding MSB 1101_2 LSB $\rightarrow 13_{10}$

LSB MSB
$\underline{A}=[1011]$

$\underset{1\times4}{\underline{A}}\cdot\underset{4\times8}{\underline{H}}=\underset{1\times8}{\vec{C}}$

$$[1011]\cdot\begin{pmatrix}1&0&0&0&0&1&1&1\\0&1&0&0&1&0&1&1\\0&0&1&0&1&1&0&1\\0&0&0&1&1&1&1&0\end{pmatrix}=(C_{01},C_{02},C_{03},C_{04},C_{05},C_{06},C_{07},C_{08})$$

C_{0x}	C_{01}	C_{02}	C_{03}	C_{04}	C_{05}	C_{06}	C_{07}	C_{08}
$A\cdot H_x$	1	0	1	1	2	3	2	2
C_{0x} MOD 2	1	0	1	1	0	1	0	0

⇓

LSB MSB
$\boxed{\vec{c}=[1\,0\,1\,1\,0\,1\,0\,0]}$

M) encoding MSB 1110_2 LSB $\rightarrow 14_{10}$

LSB MSB
$\underline{A}=[0111]$

$\underset{1\times4}{\underline{A}}\cdot\underset{4\times8}{\underline{H}}=\underset{1\times8}{\vec{C}}$

$$[0111]\cdot\begin{pmatrix}1&0&0&0&0&1&1&1\\0&1&0&0&1&0&1&1\\0&0&1&0&1&1&0&1\\0&0&0&1&1&1&1&0\end{pmatrix}=(C_{01},C_{02},C_{03},C_{04},C_{05},C_{06},C_{07},C_{08})$$

C_{0x}	C_{01}	C_{02}	C_{03}	C_{04}	C_{05}	C_{06}	C_{07}	C_{08}
$A\cdot H_x$	0	1	1	1	3	2	2	2
C_{0x} MOD 2	0	1	1	1	1	0	0	0

⇓

LSB MSB
$\boxed{\vec{c}=[0\,1\,1\,1\,1\,0\,0\,0]}$

M) encoding MSB 1111_2 LSB $\rightarrow 15_{10}$

LSB MSB
$\underline{A}=[1111]$

$\underset{1\times4}{\underline{A}}\cdot\underset{4\times8}{\underline{H}}=\underset{1\times8}{\vec{C}}$

$$[1111]\cdot\begin{pmatrix}1&0&0&0&0&1&1&1\\0&1&0&0&1&0&1&1\\0&0&1&0&1&1&0&1\\0&0&0&1&1&1&1&0\end{pmatrix}=(C_{01},C_{02},C_{03},C_{04},C_{05},C_{06},C_{07},C_{08})$$

C_{0x}	C_{01}	C_{02}	C_{03}	C_{04}	C_{05}	C_{06}	C_{07}	C_{08}
$A\cdot H_x$	1	1	1	1	3	3	3	3
C_{0x} MOD 2	1	1	1	1	1	1	1	1

⇓

LSB MSB
$\boxed{\vec{c}=[1\,1\,1\,1\,1\,1\,1\,1]}$

2. Decode the following codes. If it contains an error, explain how you can correct it; however, some errors cannot be corrected.
   (a) $1110\ 0011_2$
   (b) $1101\ 1000_2$

$$H^T=\begin{pmatrix}0&1&1&1\\1&0&1&1\\1&1&0&1\\1&1&1&0\\1&0&0&0\\0&1&0&0\\0&0&1&0\\0&0&0&1\end{pmatrix}$$ ← transpose of H

a) Decode $1110\ 0011_2$

$\vec{c}=1100\ 1110$

$$\underset{1\times4}{\vec{e}}=\underset{1\times8}{\vec{c}}\cdot\underset{8\times4}{H^T}=(1110\ 0011)\cdot\begin{array}{c}\begin{pmatrix}0&1&1&1\\1&0&1&1\\1&1&0&1\\1&1&1&0\\1&0&0&0\\0&1&0&0\\0&0&1&0\\0&0&0&1\end{pmatrix}\end{array}\begin{array}{c}0\\1\\2\\3\\4\\5\\6\\7\end{array}$$

$\vec{e}_{0x}$	$\vec{e}_{00}$	$\vec{e}_{01}$	$\vec{e}_{02}$	$\vec{e}_{03}$
	1	2	3	3
$\vec{e}_{0x}$ MOD 2	1	0	1	1

↳ matches row 1 in $H^T \rightarrow N=1$

	0	1	2	3	4	5	6	7
origional	1	1	1	0	0	0	1	1
flipped row N	1	0	1	0	0	0	1	1

$\vec{c}=[1\,0\,1\,0\,0\,0\,1\,1]$

$\vec{m}=[0\,0\,1\,1]$

$\boxed{\text{origional message: } 1100_2=12_{10}}$

b) Decode $1101\ 1000_2=\vec{c}$

$\vec{c}=[0001\ 1011]$

$$\underset{1\times4}{\vec{e}}=\underset{1\times8}{\vec{c}}\cdot\underset{8\times4}{H^T}=(0001\ 1011)\cdot\begin{pmatrix}0&1&1&1\\1&0&1&1\\1&1&0&1\\1&1&1&0\\1&0&0&0\\0&1&0&0\\0&0&1&0\\0&0&0&1\end{pmatrix}\begin{array}{c}0\\1\\2\\3\\4\\5\\6\\7\end{array}$$

$\vec{e}_{0x}$	$\vec{e}_{00}$	$\vec{e}_{01}$	$\vec{e}_{02}$	$\vec{e}_{03}$
	2	1	2	1
$\vec{e}_{0x}$ MOD 2	0	1	0	1

$\rightarrow$ $\boxed{\text{HAM-ERR}}$

3. Complete the rest of the look-up table shown below.

0	0
1	4
…	…
15	HAM_ERR

int lookup[16]: {

0	← Never lookup 0 bc 0 has no error syndrome
1	→ 4 → flip 4th bit
2	→ 5
3	HAM-ERR //001
4	6
5	H-E
6	H-E
7	3
8	7
9	H-E
10	H-E
11	2
12	H-E
13	1
14	0
15	HAM-ERR

Index Bin to Dec

$$\begin{pmatrix}0&1&1&1\\1&0&1&1\\1&1&0&1\\1&1&1&0\\1&0&0&0\\0&1&0&0\\0&0&1&0\\0&0&0&1\end{pmatrix}\begin{array}{c}0\\1\\2\\3\\4\\5\\6\\7\end{array}\begin{pmatrix}1&1&1&0\\1&1&0&1\\1&0&1&1\\0&1&1&1\\0&0&0&1\\0&0&1&0\\0&1&0&0\\1&0&0&0\end{pmatrix}\begin{array}{c}14\\13\\11\\7\\1\\2\\4\\8\end{array}$$

answer ← $\vec{e}$

Use table to calculate

lookup[13] == 1 → flip the first bit

Marian Zlateva
mzlateva@ucsc.edu
2/07/2021

CSE13s Fall 2020
Assignment 4: Hamming Codes

Description:

In this lab, I wrote up a decoder and an encoder that encodes and decodes files with optional bit noise given from an error.c source file.

Flags for both generator.c and decoder.c:

-i:	input file
-o:	output file

Files:

- generator.c
  - contains implementation of the Hamming Code generator
- decoder.c
  - contains implementation of the Hamming Code decoder
- bm.c
  - contains implementation of the bit matrix ADT
- hamming.c
  - contains implementation of the Hamming Code module
- error.c
  - adds noise to bits

Functions:

generator.c

main(int argc,char **argv)

- reads through the user's inputs
- checks to see if infile is not a valid file
- make output file permission same as input file permission
- get permission status of infile
- set permission status of outfile to infile's permission status
- initialize the generator and parity matrices
- loop through text and encodes it to the outfile
  - separate byte into nibbles
  - generate codes
  - output code
- remove garbage

high_nibble

- returns the high nibble of a byte

- makes all numbers in lower nibble 0, and shifts bytes to the right by 4

low_nibble

- returns the low nibble of a byte

- makes all numbers in the upper nibble 0

decoder.c

main(int argc,char **argv)

- ➔ reads through the user's inputs
- ➔ make output file permission same as input file permission
- ➔ get permission status of infile
- ➔ set permission status of outfile to infile's permission status
- ➔ initialize the generator and parity matrices
- ➔ initialize tracker variables
- ➔ loops through encoded text and decodes it to the outfile
  - ➝ generate text from codes
  - ➝ join nibbles
  - ➝ output result
- ➔ print error statistics
- ➔ remove garbage

low_nibble is same as generator.c

byte_from_nibble(uint8_t byte1,uint8_t byte2)

- ➢ returns the amount of bytes are needed for the specified amount of bits

- ➔ if bits == 0
  - ➝ return 0
- ➔ else if (bits % 8 == 0)
  - ➝ return bits/8
- ➔ else
  - ➝ return bits/8 +1

bm.c

BitMat *bm_create(uint32_t rows, uint32_t cols)

➢

- ➔ allocate memory for a BitMat ADT using calloc
- ➔ m->rows = rows
- ➔ m->cols = cols
- ➔ allocate memory for m->mat rows
    - ➙ size of each row is (u8 *)
- ➔ allocate memory for each m->mat col
- ➔ u32 bytesInEachRow = bytes(cols)
- ➔ for(int r=0; r<rows; r++)
    - ➙ m->mat[r] = (u8 *)calloc(bytesInEachRow, sizeof(u8))

uint32_t bytes(uint32_t bits)

➢ returns the amount of bytes are needed for the specified amount of bits

- ➔ if bits == 0
    - ➙ return 0
- ➔ else if (bits % 8 == 0)
    - ➙ return bits/8
- ➔ else
    - ➙ return bits/8 +1

void bm_delete(BitMat **m)

➢ deletes the BitMat ADT

- ➔ free each row of m->mat
- ➔ free m->mat
- ➔ free BitMat

uint32_t bm_rows(BitMat *m)

➢ returns amount of rows in BitMat

- ➔ return m->rows

uint32_t bm_cols(BitMat *m)

➢ returns amount of cols in BitMat

- ➔ return m->cols

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

- sets a bit in m->mat to 1

> - colByte = col/8
>   - get the index of the byte that the bit we want lays in
> - byte = m->matrix[row][colByte]
>   - get the byte
> - col = col%8
>   - get bit we want to modify in range [0-7]
> - mask = 1<<col
>   - make a byte with a 1 shifted c spots in (put a 1 at spot c)
> - m->mat[row][colByte] = byte | mask
>   - | means or
>   - set the byte in the ADT to the union of 'byte' and 'mask'

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

- sets a bit in m->mat to 0

> - colByte = c/8;
>   - get the index of the byte that the bit we want lays in
> - c = c%8;
>   - get the index of the bit we want in that byte
> - mask = ~(1<<c);
>   - make a byte with all 1s except one 0 at spot c
> - m->mat[r][colByte] = byte & mask
>   - multiply the 'mask' and 'byte' bytes together so that all bits that aren't in spot c stay the same while spot c turns to a 0

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

- returns the bit in the specified row and col

> - colByte = col/8
>   - get the index of the byte that the bit we want lays in
> - byte = m->matrix[row][colByte]
>   - get the byte
> - col = col%8
>   - get the index of the bit we want in that byte
> - mask = 1<<col
>   - make a byte with a 1 shifted c spots in (put a 1 at spot c)
> - uint8 result = byte & mask
>   - multiply the 'mask' and 'byte' bytes together so that all bits that aren't in spot c must be 0s
> - result = result >> index
>   - move the bit to the least significant bit spot in the byte

➔ return result

void bm_print(BitMat *m)

- ➢ sets a bit in m->mat to 1

➔ loop through the matrix and print each byte using get bit

hamming.c

int ham_rc ham_init(void)

- initializes matrices g and h
- generator
  - $$G = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 1 & 1 & 1 \\ 0 & 1 & 0 & 0 & 1 & 0 & 1 & 1 \\ 0 & 0 & 1 & 0 & 1 & 1 & 0 & 1 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 & 0 \end{pmatrix}.$$
- parity
  - $$H = \begin{pmatrix} 0 & 1 & 1 & 1 & 1 & 0 & 0 & 0 \\ 1 & 0 & 1 & 1 & 0 & 1 & 0 & 0 \\ 1 & 1 & 0 & 1 & 0 & 0 & 1 & 0 \\ 1 & 1 & 1 & 0 & 0 & 0 & 0 & 1 \end{pmatrix}.$$

- initialize matrix G
  - generator = bm_create(4,8)
  - for(int i=0; i<4; i++)
    - bm_setbit(generator,i,i)
  - for(int r=0; r<4; r++)
    - for(int c=4; c<8; c++)
      - if(r==(c%4))
        - continue
      - bm_setbit(generator,r,r,c)
- initialize matrix H
  - for(int r=0; r<4; r++)
    - for(int c=0; c<4; c++)
      - if(r==c)
        - continue
      - bm_setbit(generator,r,r,c)
  - for(int i=0; i<4; i++)
    - bm_setbit(generator,i,i+4)

ham_destroy(void)

- frees the memory taken by the matricies

- call bm_delete for generator and parity

get_bit(uint8_t byte, uint32_t c)

- gets a bit from a byte

- make a byte with a 1 shifted c spots in (put a 1 at spot c)
- multiply the 'mask' and 'byte' bytes together so that all bits that aren't in spot

c must be 0s
- ➔ move the bit to the least significant bit spot in the byte

set_bit(uint8_t *byte, uint32_t c)
- ➢ sets a bit in a byte to 1

- ➔ check if index is in range of a byte
- ➔ make a byte with a 1 shifted c spots in (put a 1 at spot c)
- ➔ set the byte in the ADT to the union of 'byte' and 'mask'

clr_bit(uint8_t *byte, uint32_t c)
- ➢ sets a bit in a byte to 0

- ➔ check if index is in range of a byte
- ➔ make a byte with all 1s except one 0 at spot c
- ➔ multiply the 'mask' and 'byte' bytes together so that all bits that aren't in spot c stay the same while spot c turns to a 0

print_bits(uint8_t byte)
- ➢ sets a bit in a byte to 0

- ➔ loop through bits and print them

multiply_mats(uint8_t a, BitMatrix *b)
- ➢ sets a bit in a byte to 0

- ➔ iterates through all the rows in the byte which is 1
  - → iterates through all the cols of b
    - ⇀ multiplies each col of a in its indexed row by each row of b in its indexed col and adds the results together
    - ⇀ puts sets sum to 0 or 1
    - ⇀ sets the bit in the result

ham_rc ham_encode(uint8_t data, uint8_t *code)
- ➢ encodes a nibble of data

- ➔ checks if init was called before and if code pointer was valid
- ➔ encodes by multiplying data with the generator matrix

ham_decode(uint8_t error)

- decodes an encoded nibble given the coded byte

→ checks if init was called before and if data pointer was valid → set data to code, assumes no bit flip is needed → multiply code with parity matrix to check for errors in bits → no error means no bit needs to be flipped → find the error bit → flip the error bit

Truth Tables

AND	0	1
0	0	0
1	0	1

OR	0	1
0	0	1
1	1	1

XOR	0	1
0	0	1
1	1	0

Not	
0	1
1	0

LOGIC

Setting a bit

we want to set a 1 in the 6th pos

```
           7 6 5 4   3 2 1 0
byte = MSB 1 0 1 1   1 0 0 1 LSB
mask = 1 << 6 (get 1 to index 6)

mask = MSB 0 1 0 0   0 0 0 0 LSB
           7 6 5 4   3 2 1 0
byte = MSB 1 0 1 1   1 0 0 1 LSB
             ↓ OR
result = MSB 1 1 1 1   1 0 0 1 LSB
```

Clearing a bit

We want to set the 6th bit to 0

```
           7 6 5 4   3 2 1 0
byte = MSB 1 1 1 1   1 0 0 1 LSB
mask = 1 << 6
mask = MSB 0 1 0 0   0 0 0 0 LSB
mask = ~(1<<6)     ← Not
mask = MSB 1 0 1 1   1 1 1 1 LSB

result = byte AND mask

           7 6 5 4   3 2 1 0
byte = MSB 1 1 1 1   1 0 0 1 LSB
mask = MSB 1 0 1 1   1 1 1 1 LSB
             ↓ AND
result = MSB 1 0 1 1   1 0 0 1 LSB
```

Getting a bit

We want to get the 4th bit

```
           7 6 5 4   3 2 1 0
byte = MSB 1 1 1 1   1 0 0 1 LSB
mask = 1 << 4   (puts a 1 shifted 4 spaces in)
mask = MSB 0 0 0 1   0 0 0 0 LSB

result = byte AND mask

           7 6 5 4   3 2 1 0
byte = MSB 1 1 1 1   1 0 0 1 LSB
mask = MSB 0 0 0 1   0 0 0 0 LSB
             ↓ AND
result = MSB 0 0 0 1   0 0 0 0 LSB  ← the bit needs to be moved to the LSB pos
result = result >> 4   (shifts result to the right by 4)
result = MSB 0 0 0 0   0 0 0 1 LSB
```

Phsudo code

set bit

```
uint8 set bit(uint8 byte, index):
    index = index % 8   (restrict index [0-7])
    mask = 1 << index
    result = byte | mask     ← or
    return result
```

clear bit

```
uint8 clrbit(uint8 byte, index)
    index = index % 8
    mask = ~(1<<index)
    resut = byte & mask
    return result
```

get bit

```
uint8 getbit(uint8 byte, index)
    index = index % 8
    mask = 1 << index
    result = byte & mask
    result = result >> index
    return resut
```

bit matrix

```
{MSB 0000 0000 LSB}
{MSB 0000 0000 LSB}
{MSB 0000 0000 LSB}
{MSB 0000 0000 LSB}
```

```
struct BitMatrix {
    U32 rows;
    U32 cols;
    U8 ** mat;

U32 bytes(U32 bits){      ← gets # of bytes needed for bits to fit in
    if(bits == 0){
        return 1
    }
    if (bits % 8 == 0){
        return bits/8
    }else{                 ← this is an int so no decimals
        return bits/8 + 1

bm_create(rows, cols)
    BitMat *m = calloc(1, sizeof(BitMat))
    m->rows = rows
    m->cols = cols
    m->mat = (BitMat **)calloc(rows, sizeof(U8 *))   ← allocate memory for each row
    for (U32 r=0; r<rows; r++)                         ← every col is a byte
        m->mat[r] = (U8 *)calloc(bytes(cols), sizeof(U8))
```

Ex

```
bm_create(4,8)
rows = 4
cols = 8 → 2 bytes
      1 byte      1 byte
0 [0000 0000   0000 0000
1 [0000 0000   0000 0000
2 [0000 0000   0000 0000
3 [0000 0000   0000 0000
```

Lab section code example

```
bm_set_bit(BitMat *m, row, col)   look at setting a bit in Logic for more info
    byte = m->matrix[row][col/8]   ← get the byte we want to modify

    col = col % 8   ← get bit in col we want to modify in range [0-7]
    mask = 1 << col
    m->mat[row][col/8] = byte | mask
    return;
```

write these in order for testing
- bm_set_bit()
- bm_get_bit()
- bm_print()

hamming.c

```
static BitMat *generator;
static BitMat *parity;

ham_rc ham_init(void){
```

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

```
generator = bm_create(4,8)

bm_set_bit(generator, 0,0)
bm_set_bit(generator, 1,1)
bm_set_bit(generator, 2,2)
bm_set_bit(generator, 3,3)
bm_set_bit(generator, 0,5)
bm_set_bit(generator, 0,6)
        ⋮
```

it looks like we can set these bits with for loops

```
first part:
for (int r = 0; r < 4; r++){
    bm_set_bit(generator, r, r);
}
second part:
for (int r = 0; r < 4; r++){
    for (int c = 4; c < 8; c++){
        if (r == c - 4){
            continue;
        }
        bm_set_bit(generator, r, c);
    }
}

bm_print(generator)   //for debugging

void bm_print(BitMat *m){
    for (int r = 0; r < m->rows; r++){
        for (int c = 0; c < m->cols; c++){
            print bm_get_bit(m, r, c)
        }
    }
    return;
}
```

Hamming Codes

$$A_{m \times n} \cdot B_{n \times p} = C_{m \times p} \qquad 0 \le i \le m, \; 0 \le j \le p$$

$$C_{ij} = \sum_{k=0}^{n} a_{ik} b_{kj} \qquad 0 \le k \le n$$

```
for (int i = 0; i < m; i++){
    for (int j = 0; j < p; j++){
        for (int k = 0; k < n; k++){
            C[i][j] = A[i][k] * B[k][j];
        }
    }
}
```

Hamming for (1100) = A

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

$$\underset{1\times 4}{AG = (0011)} \cdot \underset{4\times 8}{\begin{pmatrix} 1&0&0&0&0&1&1&1\\ 0&1&0&0&1&0&1&1\\ 0&0&1&0&1&1&0&1\\ 0&0&0&1&1&1&1&0 \end{pmatrix}} = \underset{1\times 8}{(C_{00}, C_{01}, C_{02}, C_{03}, C_{04}, C_{05}, C_{06}, C_{07})}$$

$C_{00} = (0\cdot1)+(0\cdot0)+(1\cdot0)+(1\cdot0) = 0$

$C_{01} = (0\cdot0)+(0\cdot1)+(1\cdot0)+(1\cdot0) = 0$

$C_{02} = (0\cdot0)+(0\cdot0)+(1\cdot1)+(1\cdot0) = 1$ keep val btwn 0 & 1

$C_{03} = (0\cdot0)+(0\cdot1)+(1\cdot1)+(1\cdot1) = 2 \% 2 = 0$

$\vdots$

C_{07}

$C = (00110011)$

General	Bits
multiply %2	And
add %2	XOR

← Now you can replace the stuff on top with these

error checking

$\vec{e} = \vec{c} H^T$ — parity check; error syndrome; Hamming code

$$H^T = \begin{pmatrix} 0&1&1&1\\ 1&0&1&1\\ 1&1&0&1\\ 1&1&1&0\\ 1&0&0&0\\ 0&1&0&0\\ 0&0&1&0\\ 0&0&0&1 \end{pmatrix} \leftarrow \text{transpose of } H$$

$$\vec{e} = \vec{c} H^T = (00110011) \cdot \begin{array}{c} \text{LSB} \quad H^T \quad \text{MSB} \\ \begin{pmatrix} 0&1&1&1\\ 1&0&1&1\\ 1&1&0&1\\ 1&1&1&0\\ 1&0&0&0\\ 0&1&0&0\\ 0&0&1&0\\ 0&0&0&1 \end{pmatrix} \end{array} \begin{array}{c} \\ 0\\1\\2\\3\\4\\5\\6\\7 \end{array}$$

$\vec{e} = (1011) \leftrightarrow 1101_2 = 13$ base 2 (binary)

= row 1 of H^T

```
int lookup[16]: {
 0 |            ← Never lookup 0 bc 0 has no error syndrome
 1 | → 4        → flip 4th bit
 2 | → 5
 3 | HAM-ERR //001
 4 | 6
 5 | H-E
 6 | H-E
 7 | 3
 8 | 7
 9 | H-E
10 | H-E
11 | 2
12 | H-E
13 | 1
14 | 0
15 | HAM-ERR
   ↑ Use table to calculate
```

lookup[13] ⇒ return the row in H^T

$\vec{e} = (1011)$ matches row 1 in H^T

$$\begin{array}{c} \text{Index} \\ \begin{pmatrix} 0&1&1&1\\ 1&0&1&1\\ 1&1&0&1\\ 1&1&1&0\\ 1&0&0&0\\ 0&1&0&0\\ 0&0&1&0\\ 0&0&0&1 \end{pmatrix} \begin{array}{c} 0\\1\\2\\3\\4\\5\\6\\7 \end{array} \end{array} \begin{array}{c} \text{Bin to Dec} \\ \begin{pmatrix} 1&1&1&0\\ 1&1&0&1\\ 1&0&1&1\\ 0&1&1&1\\ 0&0&0&1\\ 0&0&1&0\\ 0&1&0&0\\ 1&0&0&0 \end{pmatrix} \begin{array}{c} 14\\13\\11\\7\\1\\2\\4\\8 \end{array} \end{array}$$

answer ← $\vec{e}$

lookup[$\vec{e}$] ⇒ row in H^T

$\vec{e} = 1011$ matches row 1 in H^T → N = 1

	0	1	2	3	4	5	6	7
$\vec{c}$ origional	0	0	1	1	0	0	1	1
$\vec{c}$ flipped row N	0	1	1	1	0	0	1	1

00110011

$\vec{c} = [0111 \; 0011]$

$\vec{m} = [0011]$

$\boxed{\text{origional message: } 1100_2 = 12_{10}}$ ← Same as start!