Caterin Duarte Reynosa
cduarter@ucsc.edu

CSE13S Fall 2020
Assignment 4: Hammin
Design Document

Goal

The goal for this assignment is to create a program that encodes/decodes a hamming code as well as correct errors that are found. We will be using an (8, 4) Hamming code to encode and decode messages read from files parsed in the I/O command line. No only will this assignment focus on using bits, logical bit operations, but we will also need to know about UNIX file permissions to read and write into our files.

PreLab questions

PreLab: question #1

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

0.) 0000 = 00000000

[0][0]= (0x1)+(0x0)+(0x0)+(0x0) = 0
[0][1]= (0x0)+(0x1)+(0x0)+(0x0) = 0
[0][2]= (0x0)+(0x0)+(0x1)+(0x0) = 0
[0][3]= (0x0)+(0x0)+(0x0)+(0x1) = 0
[0][4]= (0x0)+(0x1)+(0x1)+(0x1) = 0
[0][5]= (0x1)+(0x0)+(0x1)+(0x1) = 0
[0][6]= (0x1)+(0x1)+(0x0)+(0x1) = 0
[0][7]= (0x1)+(0x1)+(0x1)+(0x0) = 0

1.) 0001 = 00011110

[0][0]= (0x1)+(0x0)+(0x0)+(1x0) = 0
[0][1]= (0x0)+(0x0)+(0x0)+(1x0) = 0
[0][2]= (0x0)+(0x0)+(0x1)+(1x0) = 0
[0][3]= (0x0)+(0x0)+(0x0)+(1x1) = 1
[0][4]= (0x0)+(0x1)+(0x1)+(1x1) = 1
[0][5]= (0x1)+(0x0)+(0x1)+(1x1) = 1
[0][6]= (0x1)+(0x1)+(0x0)+(1x1) = 1
[0][7]= (0x1)+(0x1)+(0x1)+(1x0) = 0

2.) 0010 = 00101101

[0][0]= (0x1)+(0x0)+(1x0)+(0x0) = 0
[0][1]= (0x0)+(0x1)+(1x0)+(0x0) = 0
[0][2]= (0x0)+(0x0)+(1x1)+(0x0) = 1
[0][3]= (0x0)+(0x0)+(1x0)+(0x1) = 0
[0][4]= (0x0)+(0x1)+(1x1)+(0x1) = 1
[0][5]= (0x1)+(0x0)+(1x1)+(0x1) = 1
[0][6]= (0x1)+(0x1)+(1x0)+(0x1) = 0
[0][7]= (0x1)+(0x1)+(1x1)+(0x0) = 1

3.)0011 = 00112211 %2 -> 00110011

[0][0] = (0x1)+(0x0)+(1x0)+(1x0) = 0
[0][1] =(0x0)+(0x1)+(1x0)+(1x0)= 0
[0][2]= (0x0)+(0x0)+(1x1)+(1x0) = 1
[0][3]= (0x0)+(0x0)+(1x0)+(1x1) = 1
[0][4] =(0x0)+(0x1)+(1x1)+(1x1)= 2 % 2 = 0
[0][5] =(0x1)+(0x0)+(1x1)+(1x1)= 2%2 = 0
[0][6] = (0x1)+(0x1)+(1x0)+(1x1)= 1
[0][7] =(0x1)+(0x1)+(1x1)+(1x0) = 1

4.) 0100 = 01001011

[0][0] = 0
[0][1] = 1
[0][2] = 0
[0][3] = 0
[0][4] = 1
[0][5] = 0
[0][6] = 1

5.) 0101 = 01010101

[0][0] = 0
[0][1] = 1
[0][2] = 0
[0][3] = 1
[0][4] = 0
[0][5] = 1
[0][6] = 0

[0][7] = 1

6.) 0110 = 01100110

[0][0] = 0
[0][1] = 1
[0][2] = 1
[0][3] = 0
[0][4] = 0
[0][5] = 1
[0][6] = 1
[0][7] = 0

8.) 1000 = 10000111

[0][0] = 1
[0][1] = 0
[0][2] = 0
[0][3] = 0
[0][4] = 0
[0][5] = 1
[0][6] = 1
[0][7] = 1

10.) 1010 = 10101010

[0][0] = 1
[0][1] = 0
[0][2] = 1
[0][3] = 0
[0][4] = 1
[0][5] = 0
[0][6] = 1
[0][7] = 0

12.) 1100 = 11001100

[0][0] = 1
[0][1] = 1
[0][2] = 0
[0][3] = 0
[0][4] = 1
[0][5] = 1
[0][6] = 0
[0][7] = 0

14.) 1110 = 11100001

[0][0] = 1
[0][1] = 1
[0][2] = 1
[0][3] = 0
[0][4] = 0
[0][5] = 0

[0][7] = 1

7.) 0111 = 01111000

[0][0] =0
[0][1] = 1
[0][2] = 1
[0][3] = 1
[0][4] = 1
[0][5] = 0
[0][6] = 0
[0][7] = 0

9.) 1001 = 10011001

[0][0] = 1
[0][1] = 0
[0][2] = 0
[0][3] = 1
[0][4] = 1
[0][5] = 0
[0][6] = 0
[0][7] = 1

11.) 1011 = 10110100

[0][0] = 1
[0][1] = 0
[0][2] = 1
[0][3] = 1
[0][4] = 0
[0][5] = 1
[0][6] = 0
[0][7] = 0

13.) 1101 = 11010010

[0][0] = 1
[0][1] = 1
[0][2] = 0
[0][3] = 1
[0][4] = 0
[0][5] = 0
[0][6] = 1
[0][7] = 0

15.) 1111 = 11110000

[0][0] = 1
[0][1] = 1
[0][2] = 1
[0][3] = 1
[0][4] = 0
[0][5] = 0

[0][6] = 0
[0][7] = 1

[0][6] = 0
[0][7] = 0

PreLab: Number 2

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

a.) 1100 0111 =1011 (reversed the endianness)

[0][1]= (1x0) + (1x1) + (0 x1) + (0x1) + (0x1) + (1x0) + (1x0) + (1x0) = 1
[0][2]= (1x1) + (1 x0) + (0 x1) + (0 x1) + (0x0) + (1 x1) + (1x0) + (1 x0) = 2 % 2 = 0
[0][3]= (1 x1) + (1 x1) + (0 x0) + (0 x1) + (0 x0) + (1 x0) + (1x1) + (1x0) = 3 % 2 = 1
[0][4]= (1x1) + (1 x1) + (0 x1) + (0 x0) + (0 x0) + (1x0) + (1 x0) + (1x1) = 3% 2 = 1
Error in the 2nd row, to fix this we need to flip the 2nd bit of the code, thus getting us 1000 0111

b.) 1101 1000 = 1010 (not reversed)
0001 1011 = 0101 (reversed the endianness)

[0][1]= (0x0) + (0x1) + (0x1) + (1x1) + (1x1) + (0x0) + (1x0) + (1x0) = 2 % 2 = 0
[0][2]= (0x1) + (0 x0) + (0 x1) + (1 x1) + (1x0) + (0 x1) + (1x0) + (1 x0) = 1
[0][3]= (0 x1) + (0 x1) + (0 x0) + (1 x1) + (1 x0) + (0 x0) + (1x1) + (1x0) = 2 % 2 = 0
[0][4]= (0x1) + (0 x1) + (0 x1) + (1 x0) + (1 x0) + (0x0) + (1x0) + (1x1) = 1
This produces multiple errors, thus we cannot correct them

I can check if a code has an error if the code is part of the lookup table. It can be corrected because the number on the look up table is the index of the code’s error. If the code is not in the lookup table, it means there are two errors and we can only correct one.

PreLab: Number 3

Look-up table

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

Summary + Rules

1.) Parse command-line options (-i and -o) to read from a file and to write to a file
2.) Allocate memory for the matrixes
3.) Create a hamming code
4.) Iterate through the -i: input file
5.) Encode the messages in the file
6.) Decode errors
7.) Free the memory
8.) Code the files
9.) The decoding file will be similar to this

Function/Goal of each file

File I need to create:

- hamming .c
  - Will be creating the hamming (8,4) code, the G/H Bit matrices, encoding messages, and decoding messages
- Bm.c
  - Will be creating bit matrices (a 2-d array of bits) allocating memory for the G/H matrices, settings the rows, setting the cols, setting bits, clearing bits, and getting the bits
- generator .c
  - Reading the commands (-i -o) parsed through the command line, initializing the hamming code, reading from the file, encode the messages
- Decoder.c
  - Reading the commands (-i -o) parsed through the command line, initializing the hamming code, reading two bytes from the file (lower + upper nibble) , decode the message, return stats about how many bytes were processed, how many errors corrected, how many errors were not corrected, and the error rate.
- MakeFile
  - target : Dependencies
  - Gen: goal: build generator.c, Dependencies: hamming.c hamming.h bm.c bm.h, generator.c
  - Dec: goal: build decoder.c, Dependencies: hamming.c hamming.h bm.c bm.h, decoder.c
  - Err: goal: build error.c, Dependencies: error.c

Pseudocode

- Hamming.c
  - Create ham_rc (given in the asgn document)
  - ham_rc ham_init(void)
    - Create the G and H matrices
    - generator = bm_create(4,8)
    - Generator = bm_set_bit(0,0)
    - Generator = bm_set_bit(0,1)

        - Generator = bm_set_bit(0,2)
        - Generator = bm_set_bit(0,3)
        - Do the same as ^^ for H matrix
            - Do a for loop if i can to look cleaner
        - Allocate memory with calloc for generator and for h
        - Return HAM_OK if they were created
        - Else return HAM_ERR
    - ham_rc ham_destroy(void)
        - Use free to free the memory allocated
    - ham_rc ham_encode(uint8_t data, uint8_t *code)
        - Generates the hamming code for the nibble of data
        - Multiply the byte by the g matrix
        - Two nested loops
        - For i in range (0, code)
            - For j in range (0, generator)
            - Multiply (or do shift if i want) [i][j]
            - If hamming code was generated, return HAM_OK
            - Else return HAM_ERR
    - ham_rc ham_decode(uint8_t code, uint8_t *data)
        - Decodes the hamming code
        - Multiply the code by the h matrix
        - Two nested loops
        - For i in range (0, code)
            - For j in range (0, h)
            - Multiply (or do shift if i want) [i][j]
            - If result of multiplication == in lookup table
                - Correct the index of the bit that corresponds to the value of the lookup table
                - Return ham_err_ok
                - Counter that counts errors corrected += 1
            - If result == (0,0,0,0)
                - Return ham_ok
            - Else
                - Return ham_err
                - Counter that counts errors not corrected += 1
- Bm.c
    - BitMat *bm_create(uint32_t rows, uint32_t cols)
        - Use Calloc to allocate memory for bitmat
        - Set the pointers
        - Use Calloc to allocate memory for pointer
        - For loop that will allocate memory for rows/cols
        - If allocation failed, return Null
        - Else return pointer
    - void bm_delete(BitMat **m)

        - Free memory allocated for the rows/cols
        - Free memory allocated for pointer
        - Free memory allocated for bitmat
    - uint32_t bm_rows(BitMat *m)
        - Return # of rows (similar to last asgn)
    - uint32_t bm_cols(BitMat *m)
        - Return # of cols (similar to last asgn)
    - void bm_set_bit(BitMat *m, uint32_t row, uint32_t col)
        - Will set the bits for the G and H matrices (set 1)
        - Byte = m->mat[row][col/8]
        - Col = col % 8
            - Will think about a more efficient thing to do later o
        - Mask = a << col
        - Return m-> mat[row][col/8] = byte ored with mask
    - void bm_clr_bit(BitMat *m, uint32_t row, uint32_t col)
        - Clears the bit at a specific location (clear to 0)
        - Byte = m->mat[row][col/8]
        - Col = col % 8
        - Mask = ~(a << col)
        - Return byte and with mask
    - void bm_get_bit(BitMat *m, uint32_t row, uint32_t col)
        - Checking to see if a bit is 0 or 1
        - Byte = m->mat[row][col/8]
        - Col = col % 8
        - Mask = ~(a << col)
        - Result = byte and with mask
        - Result = result >> byte
        - Return result
    - void bm_print(BitMat *m)
        - For i in rows
            - For i in cols
                - If bit == 1
                    - Print (“1”)
                - Else
                    - print (“0”)
- generator .c
    - include the headers
    - Set the options -i and -o
    - Use fgetc to get the data from infile
    - Do getop for -i and -o
    - ham_init
    - While the bytes of the file == 0
        - Lower nibble
        - Upper nibble

        - Generate matrix (upper + lower)
        - Fputc, print to the outfile lower nibble
        - Fputc, print to the outfile upper nibble
    - Ham_delete
    - Close files
- decoder .c
    - include the headers
    - Set the options -i and -o
    - Use fgetc to get the data from infile
    - Do getop for -i and -o
    - ham_init
    - While the bytes of the file == 0
        - Lower nibble
        - Upper nibble
        - Decode lower
        - Decode upper
        - Rebuild the message using the helper function on the asgn doc
        - Fputc, print to the message
    - Print the stats I have to print
    - Ham_delete
    - Close files