Assignment 4- Hamming Codes

This program, or more accurately— combination of programs— encodes a given input using Hamming codes into a specified output and can decode the Hamming codes into a readable format, even when errors are introduced. When decoding, it let the user know the number of bytes processed, whether there were any errors when decoding the Hamming codes and the number of fixable errors found as well as unfixable ones— where the count of unfixable was used to compute error rate.

Pre-Lab Questions:

1) Calculate Hamming codes-

Given G =

$$\begin{bmatrix} 1\ 0\ 0\ 0\ 0\ 1\ 1\ 1 \\ 0\ 1\ 0\ 0\ 1\ 0\ 1\ 1 \\ 0\ 0\ 1\ 0\ 1\ 1\ 0\ 1 \\ 0\ 0\ 0\ 1\ 1\ 1\ 1\ 0 \end{bmatrix}$$

The Hamming codes for the following nibbles can be calculated by doing: (nibble*G)%2

0000(0)= (0000*G)%2 = 00000000

0001(1)= (1000*G)%2 = 10000111

0010(2)= (0100*G)%2 = 01001011

0011(3)= (1100*G)%2 = 11001100

0100(4)= (0010*G)%2 = 00101101

0101(5)= (1010*G)%2 = 10101010

0110(6)= (0110*G)%2 = 01100110

0111(7)= (1110*G)%2 = 11100001

1000(8)= (0001*G)%2 = 00011110

1001(9)= (1001*G)%2 = 10011001

1010(10)= (0101*G)%2 = 01010101

1011(11)= (1101*G)%2 = 11010010

1100(12)= (0011*G)%2 = 00110011

1101(13)= (1011*G) = 10110100

1110(14)= (0111*G)%2 = 01111000

1111(15)= (1111*G)%2 = 11111111

2) Decode the codes-

a) (1110 0011)*(H^T) = 0100 -> matches 5th row; therefore, to fix error, flip 5th element

b) (1101 1000)*(H^T) = 0101 -> although there is an error in this code, it is an error that cannot be fixed since the nibble does not match any row value in H^T.

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	HAM_ERR
13	1
14	0

15	HAM_ERR

Basic Organization:

1) bm.c

- struct BitMatrix: specifies the members of the struct that will be used within other functions with its int type- uint8_t **mat

-*helper function (bytes(bits)): contains calculation for splitting given number of bits into bytes

-BitMat *bm_create: allocates memory to BitMat with consideration to the row-major order

- void bm_delete: frees the memory that was allocated to BitMat data type
- uint32_t bm_rows: returns rows of passed in BitMat
- uint32_t bm_cols: returns cols of passed in BitMat
- void bm_set_bit: sets specific bit by left shifting and using OR
- uint8_t bm_get_bit: gets bit value by left shifting and using AND; then right shifting
- void bm_clr_bit: clears bit by left shifting and using the NOT of that shift and AND
- void bm_print: to help debug

2) hamming.c

- ham_rc init: initializes matrices and H using matrix specifications from bm.c
- void ham_rc ham_destroy: frees any memory that was allocated to matrices from the module to prevent memory leaks -> frees keeping in mind the analogy that a bucket must be emptied before it is thrown away
- ham_rc ham_gen: uses matrix G to encode first argument data and put into pointer code
    * needs to account for failure to initialize module and NULL pointer and/or data
- ham_rc ham_dec: similar to gen but uses transposed matrix H to decode first argument code and put into pointer data

    ↳makes use of lookup table to make the process of identifying errors more efficient.

    - once lookup table is used to identify error, the error at the specific bit location will be flipped and the lower nibble is taken from the integer that was previously vector c.

    * needs to account for failure to initialize modules and NULL pointer and/or code

3) generator.c and decoder.c

- defines command-line getopt() options

  - -i: specifies input file to be read for encoding/decoding(default = stdin)
  - -o: specifies output file to be printed out to after encoding/decoding (default = stdout)

- contains main()
  - calls functions from hamming.c (including those to initialize Hamming modules)
  - uses return values from hamming.c to print output when specific getopt() options are called
  - In case of encoder.c:
    - most functions except ham_decode will be called
    - passes one byte at a time(with only the lower nibble “shown”) to ham_encode which generates Hamming code-> do for upper and lower nibble of given byte
    - Hamming code for LOWER nibble written first followed by code for UPPER nibble
  - In case of decoder.c:
    - most functions except ham_encode will be called
    - passes one byte(Hamming codes) at a time (FIRST byte read and passed from main = LOWER nibble; SECOND byte = UPPER nibble)
    - once decoded-> matches result to lookup table anc checks for error
      - in the case of fixable error: fixes bit at location and then sends main fixed code and returns HAM_ERR_OK
      - in the case of unfixable error: returns HAM_ERR
      - in the case of no error: sends to main the original code and returns HAM_OK
    - for every byte in the two bytes read at a time through main, main should take lower nibble of both and pack into one byte
    - writes reconstructed byte to output
    - repeats for all of input file
    - keeps a count of how many HAM_OK/HAM_ERR_OK/HAM_ERR were returned as well as the number of bytes read and uses count to print stats for total bytes processed, corrected errors, uncorrected errors, and error rate
  - Both decoder.c and generator.c will after these steps call ham_destroy and close the input and output files

* FOR SPECIFIED INPUT/OUTPUT FILES: OUTPUT FILE SHOULD HAVE SAME FILE PERMISSIONS AS INPUT

- fstat() retrieves an open file's permissions + fchmod() changes permissions of outfile to match infile
    - both functions expect **file descriptor** returned by low-level IO open() SO USE fileno() to get file descriptor of open stream

Pseudocode:

```
bm.c: (referenced to Eugene's code)
    bytes(bites):
        returns bytes given bits
    *bm_create:
        allocate memory to bitMat in row-major order
        after each mem-alloc, check NULL
            if condition met, return HAM_ERR
    bm_get_bit(matrix, row, col):
        normalize col vals to 0-7 and shift mask by col
        mask = 1 << modified cols
        bit = (byte & mask) >> cols
        val bit at given index in bitMatrix
        return val bit
    bm_set_bit(matrix, row, col):
        normalize cols to 0-7
        mask = 1 << cols
        set bit = byte|mask
    bm_clr_bit(matrix, row, col):
        normalize cols to 0-7
        mask = ~(1 << cols)
        clr bit = byte & mask
    bm_print(matrix):
        for i in row, < row length, i++
            for j in col, < col length, j++
                print matrix
hamming.c:
    init:
        initializes modules-> matrices G and H w/ bm_create
        checks NULL for matrix G and H
            if condition met, return HAM_ERR
            else continue
    ham_destroy:
        calls bm_delete for G and H
    Ham_gen: (referenced to Eugene’s code)
        data = byte w/ only nibble
        for a = 0; < 4; a++
            shift & mask by a
            store data in BitMatrix[0][a]
        for i = 0; < rows(BM), i++
            for j = 0; < cols(G), j++
                for k = 0; < rows(G), k++;
                product = new BM[i][k] * G[k][j] (results in eight bits)
        *code = product
        if successful, return HAM_OK
            else, (and for void pointer/module uninitialized) return HAM_ERR
    ham_dec:
        initialize lookup table as array
        for a = 0; < 8; a++
            shift & mask by a
            store data in BitMatrix[0][a]
        for i = 0; < rows(BM), i++
            for j = 0; < cols(H), j++
                for k = 0; < rows(H), k++;
                product = new BM[i][k] * H[k][j] (results in four bits)
        product = new BM * H
        lookup[product]
            if == -2: no error-> return HAM_OK
            if == -1: unfixable error-> return HAM_ERR
            if >= 0: fixable error
                find corresponding row in H matrix using lookup[product]
                fix bit at respective column in created BM
                BM-> uint8_t corrected
                *data = uint8_t corrected

encoder.c:
    set up getopt options
    open infile and outfile
    while infile != EOF:
    read first byte of input file
    ham_init
    for each byte
        lowNib = lower_nibble(byte)
        highNib = higher_nibble(byte)
        ham_encode(lowNib, &code)
        ham_encode(highNib, &code)
        if(ham_encode == HAM_OK)
            putc (write to)*code into outfile (code for LOW nibble then HIGH)
    ham_destroy
    close infile and outfile

decoder.c:
    set up getopt options
    open infile and outfile
    while infile != EOF:
    read one character of input file
        first = fgetc()
        byteCount ++
    read another character
        second = (fgetc())
        byteCount++
    ham_init
    if(ham_decode == HAM_ERR_OK || HAM_OK)
        for(m = 0; < 2; m++)
            if (m == 0)
                result1 = ham_decode(first, &data)
                firstN = lower_nibble(result 1)
            else
                result2 = ham_decode(second, &data)
                secondN = lower_nibble(result 2)
        final byte = pack_nibbles(firstN, secondN)
        putc (write to)*code into outfile
    if(ham_decode == HAM_ERR_OK)
        correctedCount ++
    else if(ham_decode == HAM_ERR)
        uncorrectCount ++
    error rate = uncorrectCount/byteCount
    ham_destroy
    close infile and outfile
```