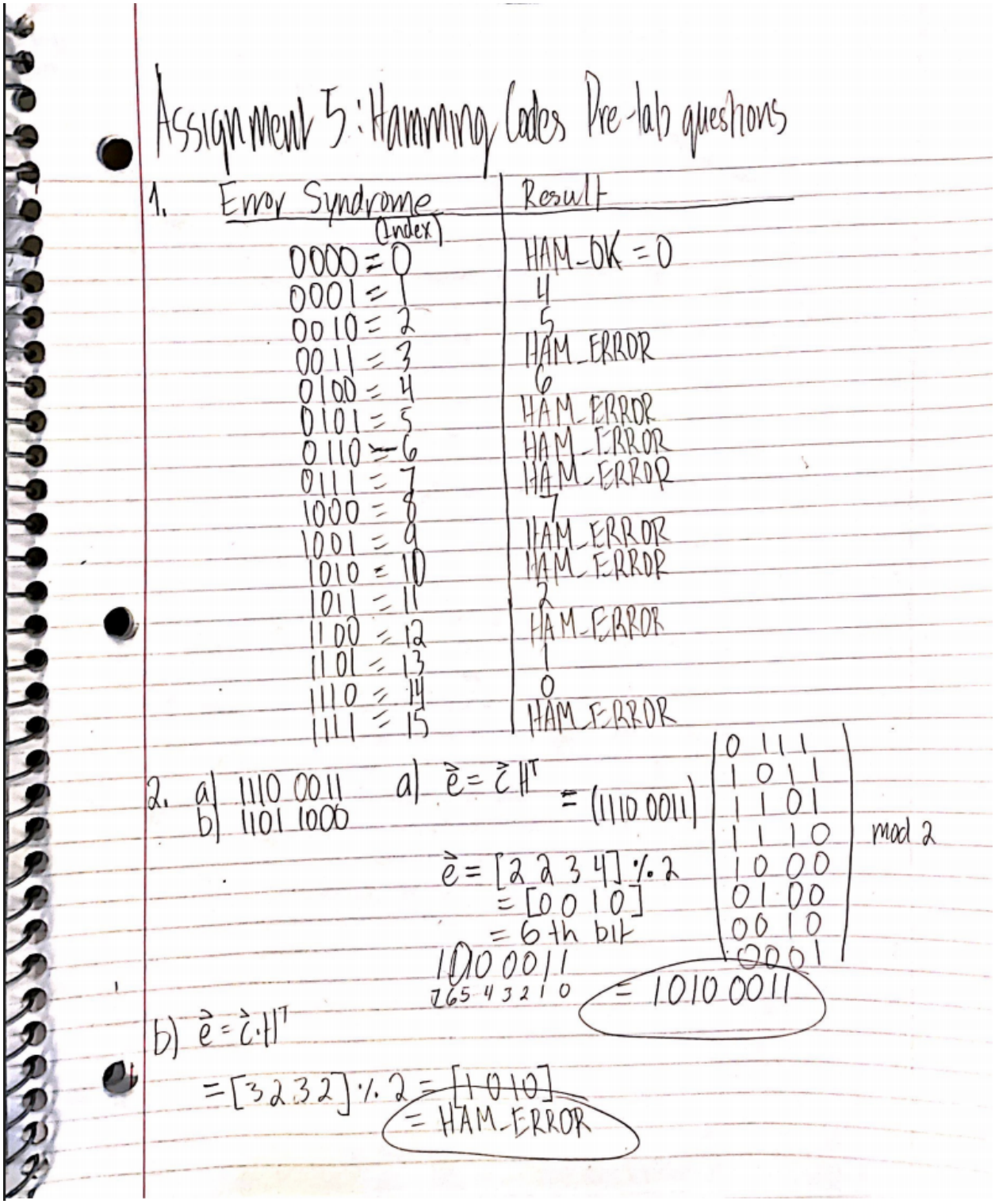
Ravjodh Heer

Assignment 5: Hamming Codes - DESIGN.pdf

Purpose:

_____The purpose of this lab is to construct (8, 4) hamming codes using bit vectors and bit matrices in order to parse and encode as well as decode messages through binary values. We will take in a message through a file and encode it using hamming (8, 4) coding techniques and output it to another file. This message will be encoded and will be secure until decoding. We will also have a decoding function that will take in an encoded message from a file and decode it in order to return the message to us. Our encode and decode will also be tested by adding noise to check how well it can correct errors and the errors will be statistically measured against the entropy value given to the function. The injected noise will mess around with the bits and make the message incorrect but if implemented correctly, our program will be able to detect and fix those error, or give a HAM_ERROR return value if the errors are too many to be fixed using the hamming procedure. We will also print statistics of how many errors the program corrected, how many it did not correct and the total bytes that it processed.

Pre-lab:



Assignment 5: Hamming Codes Pre-lab questions

1.

Error Syndrome (Index)	Result
0000 = 0	HAM_OK = 0
0001 = 1	4
0010 = 2	5
0011 = 3	HAM_ERROR
0100 = 4	6
0101 = 5	HAM_ERROR
0110 = 6	HAM_ERROR
0111 = 7	HAM_ERROR
1000 = 8	7
1001 = 9	HAM_ERROR
1010 = 10	HAM_ERROR
1011 = 11	3
1100 = 12	HAM_ERROR
1101 = 13	1
1110 = 14	0
1111 = 15	HAM_ERROR

2. a) 1110 0011
   b) 1101 1000

a) $\vec{e} = \vec{c} H^T$ = (1110 0011)

$$H^T = \begin{pmatrix} 0 & 1 & 1 & 1 \\ 1 & 0 & 1 & 1 \\ 1 & 1 & 0 & 1 \\ 1 & 1 & 1 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \text{ mod 2}$$

$\vec{e}$ = [2 2 3 4] % 2
= [0 0 1 0]
= 6th bit

1000011
7 6 5 4 3 2 1 0

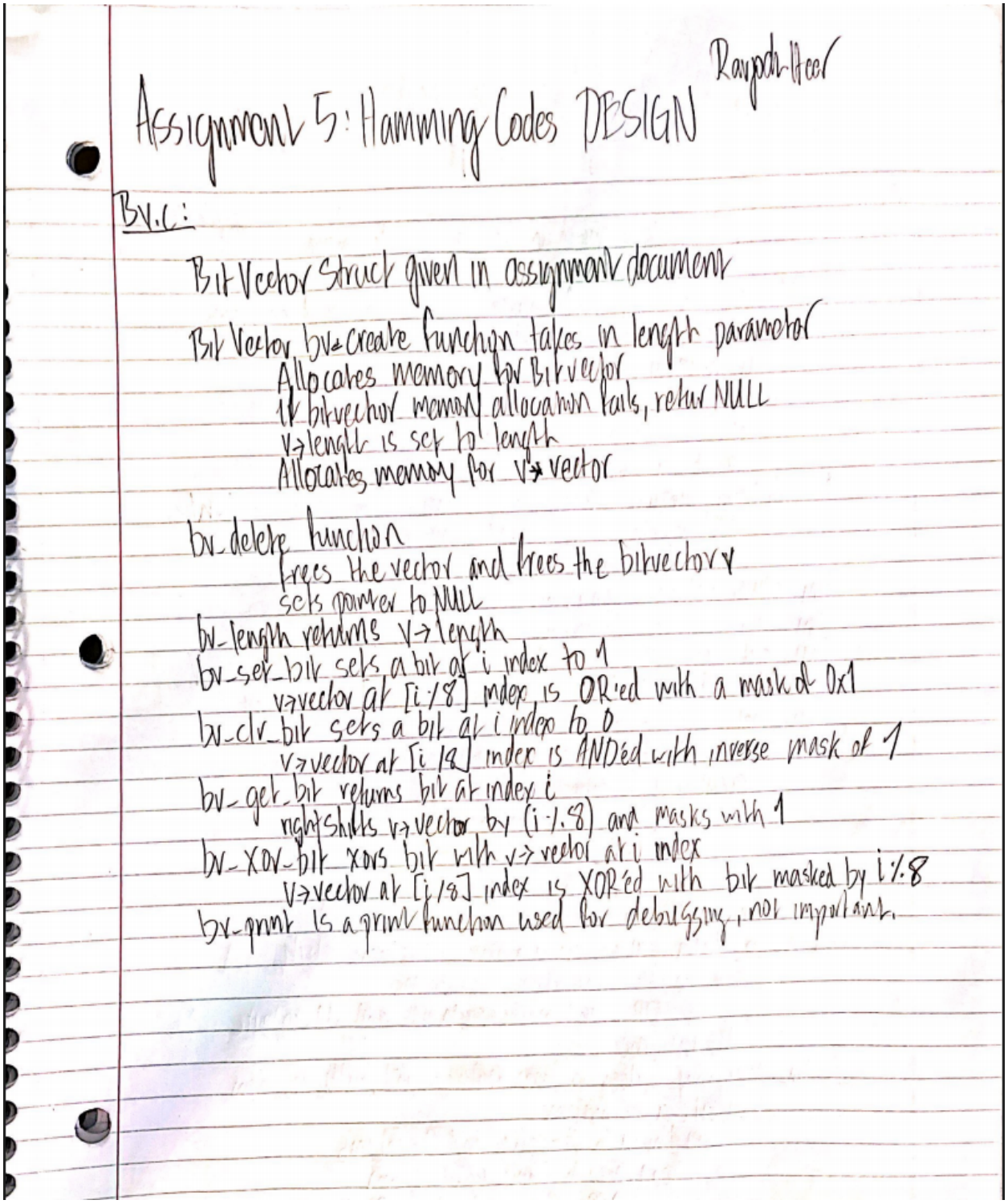
= 1010 0011

b) $\vec{e} = \vec{c} \cdot H^T$

= [3 2 3 2] % 2 = [1 0 1 0]

= HAM_ERROR

Pseudocode Draft:



Assignment 5: Hamming Codes DESIGN

Rayyan Heer

BV.c:

Bit Vector Struct given in assignment document

Bit Vector bv_create function takes in length parameter
- Allocates memory for Bitvector
- If bitvector memory allocation fails, return NULL
- v→length is set to length
- Allocates memory for v→vector

bv_delete function
- frees the vector and frees the bitvector v
- sets pointer to NULL

bv_length returns v→length

bv_set_bit sets a bit at i index to 1
- v→vector at [i/8] index is OR'ed with a mask of 0x1

bv_clr_bit sets a bit at i index to 0
- v→vector at [i/8] index is ANDed with inverse mask of 1

bv_get_bit returns bit at index i
- rightshifts v→vector by (i%8) and masks with 1

bv_XOR_bit xors bit with v→vector at i index
- V→vector at [i/8] index is XOR'ed with bit masked by i%8

bv_print is a print function used for debugging, not important.

Assignment 5: Hamming Codes DESIGN

bm.c

Struct for bitmatrix given in assignment document

BitMatrix bm_create with parameters for rows and columns
- Allocates memory for the bit matrix: m
- If memory allocation fails, return NULL
- m->rows is equal to rows
- m->cols is equal to columns
- Creates a vector using bv_create

bm_delete function deletes the matrix and frees everything
- deletes the bitvector and bitmatrix
- frees the pointers and returns NULL

bm_rows returns bm->rows

bm_cols returns bm->cols

bm_set_bit calls bv_set_bit at r * m->cols + c index

bm_clr_bit calls bv_clr_bit at r * m->cols + c index

bm_get_bit returns bv_get_bit at r * m->cols + c index

bm_from_data takes in byte and length parameters
- creates a bitmatrix of 1 * length size
- for loop that iterates through length
  - if get_bit is 1, set_bit at 0, i to 1
  - else, clr_bit at 0, i to 0
- return the bitmatrix

bm_to_data takes in a matrix and returns a byte
- for loop to iterate through columns
  - ORs and masks each bit and adds to byte
- returns the byte

bm_multiply takes in two matrices and multiplies them
- Create a bitmatrix
- loop through A->rows and B->colums
  - loop through same row|column
    - XOR value with A,i,k & Bk,j
- return result

Assignment 5: Hamming Codes DESIGN

Hamming.c

Enumerates a list of all possible hamming codes

Professor provided upper/lower nibble packing functions *

Ham_encode takes in a Bitmatrix and a message
- takes the lower nybble given and stores it
- takes the upper nibble given and stores it
- matrix multiplies Generator matrix and lower nibble
- matrix multiplies Generator matrix and upper nibble
- packs nibbles into a byte and returns it.

Ham_decode takes in a H transpose matrix, code and message
- multiplies H transpose with lower nibble
- multiplies H transpose with upper nibble
- if the code given is 0000, return HAM_OK
- Iterates over HAM_STATUS table and checks each value
  - base on the value, returns ham_error or ham_ok
  - if error detected, corrects it
    - returns ham_corrects
  - else, returns ham_status message
- returns the message and ham status

Assignment 5: Hamming Codes DESIGN

Encode.c

GetOpt function to parse command options *

Create Generator matrix G using bm_create function

For Loop {
- Use fgetc() to read bytes from specified file
- Call ham_encode function with generator matrix and msg
- Output to file output
}

Return 0, terminate program

Decode.c:

GetOpt function to parse command options *
Create H transpose matrix using bm_create
Use fgetc() function to read in both lower and upper nibbles
For loop to iterate through each byte:

call ham_decode to decode the message
return number of bytes processed
return hamming codes corrected
return hamming codes that could not be corrected

use fputc() function to send resulted message to outfile
return statistics
close infile and outfile
return 0 and terminate program

Corrections to pre-lab questions:

ERROR SYNDROME	INDEX
0111 = 7	3

correction: row 7 on the table should equate to the 3rd index, not HAM_ERROR

Pseudocode FINAL:

bv.c:

struct for BitVector given in assignment document

Code taking inspiration from my previous stack code

BitVector bv_create function with length as input parameter
  Allocates memory for BitVector
  if memory allocation fails, return NULL
  set v->length to length
  equation to calculate byte allocation for
  v->vector allocation
  allocates memory for v->vector
  if memory allocation fails, free v and return NULL
returns vector v

bv delete taking inspiration from my previous stack code

bv_delete function deletes the bitvector
  frees the pointer to v->vector and frees v and sets v to NULL

bv_length returns length pointer of v

set bit and clear bit implementation taking inspiration from Sahiti's lab section walkthrough

bv_set_bit function takes in vector and index i as input
  creates a value to store result of masking
  index value by 1
  i/8 index of vector is OR'd with mask value at index, OR preserves the value (sets bits to 1)

bv_clr_bit function takes in vector and index i as input
  creates a value to store the result of inverting the mask of index i
  i/8 index of vector is AND'ed with mask value, AND sets previous values to 0 (clears bits to 0)

bv_get_bit function takes in vector and index i as input
  modulus value storing index position is created
  mask value of 0x1 is stored in mask variable
  returns i/8 index of vector which is right shifted by modulus & mask

bv_xor_bit function takes in vector, index i and a bit
creates a value to store the result of right shifting index i
i/8 index of vector is XOR'ed with mask value, XOR preserves previous value if 0, sets to 0 if same

bv_print function for debugging reasons, not important

bm.c:

struct for BitMatrix containing rows, cols, and vector BitVector

Bitmatrix constructor function based on my previous stack code from assignment 3

BitMatrix bm_create function which takes in rows and cols as input
Allocates memory for a bitmatrix m
if memory allocation fails, return NULL
sets m->rows to rows function input
sets m->cols to cols function input
calls bv_create function to create a bitvector m->vector
if vector creation unsuccessful, free bitmatrix m and return NULL
returns bitmatrix m

destructor function bm_delete based on my previous stack destructor code from assignment 3

bm_delete function deletes the created matrix
calls bv_delete to delete the bitvector m->vector
frees bitmatrix m and sets pointer to m to NULL

bm_rows function returns bitmatrix m->rows

bm_cols function returns bitmatrix m->cols

bm_set_bit function calls bv_set_bit to set bit at rows and columns position

bm_clr_bit function calls bv_clr_bit to clear bit at rows and columns position

bm_get_bit function calls bv_get_bit to get bit at rows and columns position

bm_from_data code taking inspiration from sahiti's lab section

bm_from_data turns a byte of given length to a bitmatrix of 1 * length size
- if the given length is greater than 8, return NULL
- creates an empty bitmatrix of 1 * length size to store byte into
- iterates over the length of the inputted byte
  - masks the byte with 1 left shifted by i in order to retrieve current bit value
    - if current bit value is 1, call bm_set_bit to set the bit at current position to 1
  - otherwise calls bm_clr_bit to clear bit at current position
- returns the resulting bitmatrix

bm_to_data code taking inspiration from brian's lab section

bm_to_data takes in a bitmatrix and turns it into a byte
- introduces val variable to store returning byte
- for loop that iterates through columns of bitmatrix
  - gets bit at current i index and stores it into variable "new"
  - iteratively OR's val by new left shifted by i
- returns val variable containing new byte

bm_multiply function takes two matrices and multiplies them
- checks if the columns of first matrix and rows of second matrix are equal, if not, returns NULL
- creates a result matrix to store result into
- for loop iterates through the rows of the first matrix
  - for loop iterates through the columns of the second matrix
    - creates a sum variable to store values in
    - iterates through joint column/row of both matrices
      - iteratively XOR's sum variable with first matrix's current bit AND'ed with second matrix's current bit
    - mods sum by two to make sure result matrix only contains 0 and 1
    - if the sum value is 1, set the bit at the current position
    - otherwise clear the bit at the current position
- return the resulting matrix

bm_print function used for debugging reasons, not important

<u>hamming.c:</u>

initializes external variable uncorrected errors which keeps track of uncorrected errors
initializes external variable corrected errors which keeps track of corrected errors

Code taking inspiration from Sahiti's encode and decode explanation in lab section

creates array of bits containing ham_status codes

lower and upper nibble and byte packing code given in assignment document

ham_encode function taking in generator matrix and message as input
- creates a bitmatrix of the given message
- creates a resulting matrix of bm_multiply result between generator and message matrices
- creates a 8 bit result variable to store bm_to_data of result matrix
- deletes message matrix and result matrix to prevent memory leaks
- returns the 8 bit result

ham_decode function with ham_status return values,
- h-transpose, code input and msg output
- creates bitmatrices for code bm_from_data and matrix multiply result between h-transpose and code
- stores resulting matrix converted to byte into 8 bit error variable
- stores the ham_status code at error index to 8 bit value if the error is 0, deletes matrices to prevent memory leaks, sets msg to lower nibble of code and returns HAM_OK
- if the error is -1, sets msg to original msg input, increments uncorrected erros by 1, deletes matrices and returns HAM_ERR
- otherwise checks bit value of error bit and clears or sets it based off current value
  - stores changed matrix to 8 bit value using bm_to_data
  - takes lower nibble of decoded changed value using helper function
  - return lower nibble of changed code through msg
  - increments corrected errors external variable by 1
  - deletes matrices to prevent memory leaks and returms HAM_CORRECT

<u>encode.c:</u>

lower/upper nibble and packing/unpacking code supplied in assignment document

Getopt copied from my assignment 4 tsp.c code

main function that takes in getopt arguments
- sets infile path to stdin as default
- sets outfile path to stdout as default
- sets opt to 0 for getopt function
- getopt function while statement
  - switch statement for getopt function

case h defined as help statement
series of print statements for help statement.
case i for infile
sets infile to given optarg and opens for reading
if infile is NULL
prints error "failed to open"
closes the file
returns 1 and terminates
case o for outfile
sets outfile to given optarg and opens for writing
if outfile is NULL
prints error "failed to open"
closes infile
closes outfile
returns 1 and terminates
default case printing out error statement if user input is invalid
prints same help statement

fchmod code given in assignment document to set file permissions;

Generator Matrix creation using bm_create and bm_set_bit functions:

creates c variable to store read bytes
while loop that iterates through each byte and stores it in c variable until end of line
separates byte into lower and upper nibbles
encodes lower byte and stores in 8 bit variable
encodes upper byte and stores in 8 bit variable
outputs lower encoded byte to outfile
outputs upper encoded byte to outfile
deletes all used matrices to prevent memory leaks
Returns 0 and ends program

decode.c:

defines external stat tracking variables

lower/upper nibble and packing/unpacking code supplied in assignment document

Getopt copied from my assignment 4 tsp.c code and my assignment 5 encode.c code

main function that takes in getopt arguments

sets infile path to stdin as default
sets outfile path to stdout as default
sets verbose boolean to false as default
sets opt to 0 for getopt function
getopt function while statement
  switch statement for getopt function
  case h defined as help statement
    prints out a series of print lines for help statement
  case i for infile
    sets infile to given optarg and opens for reading
    if infile is NULL
      prints error "failed to open"
      closes the file
      returns 1 and terminates
  case o for outfile
    sets outfile to given optarg and opens for writing
    if outfile is NULL
      prints error "failed to open"
      closes infile
      closes outfile
      returns 1 and terminates
  case v for verbose
    sets verbose to true
  default case printing out error statement if user input is invalid
    series of print statements for same help function

fchmod code given in assignment document to set file permissions;

H-transpose Matrix creation using bm_create and bm_set_bit

creates a variable a to store first byte read
creates a variable b to store second byte read
while loop to read in and store 2 bytes at a time to a and b variable
  increments total_bytes_processed variable by 1

  creates 8 bit variables msg1 and msg2 to store first decoded result and second
   decoded result
  ham_decodes using h-transpose matrix, first byte and msg1 as inputs
  ham_decodes using h-transpose matrix, second byte and msg2 as inputs

  packs upper and lower bytes together and stores them in 8 bit variable

fputs the packed resulting byte to outfile
calculates the error rate as uncorrected errors / total bytes processed

if verbose is true, prints out statistics to stderr
deletes created matrices and closes infile and outfile, ends program