Aditya Agrawal
aagraw14@ucsc.edu
Spring 2021

CSE13s
Assignment 5: Hamming Codes
Design Document

This program will implement an encoder which will generate Hamming codes given input data and a decoder which will decode the generated Hamming codes. This will be accomplished through the use of ADTs called a bit vector and a bit matrix. The program will utilize bitwise operations and logical operations such as xor to change or access specific bits within a bit vector.

Pre-Lab Questions:

1. 0 | 0/HAM_OK
   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(bit index 0)
   15 | HAM_ERR
2.
   a. $1110\ 0011_2$
      c = [1 1 0 0 0 1 1 1]
      $e = c * H^T$ = [1 2 3 3] (mod 2) = [1 0 1 1] = $1\ 1\ 0\ 1_2 = 13_{10}$
      According to the lookup table the bit in index 1 needs to be flipped because the value of the error code is the matrix 1 0 1 1 which is 1 1 0 1 in binary which corresponds to error code 13 in the table.
   b. $1101\ 1000_2$
      c = [0 0 0 1 1 0 1 1]

$e = c * H^T = [2\ 1\ 2\ 1] \pmod 2 = [0\ 1\ 0\ 1] = 1\ 0\ 1\ 0_2 = 10_{10}$

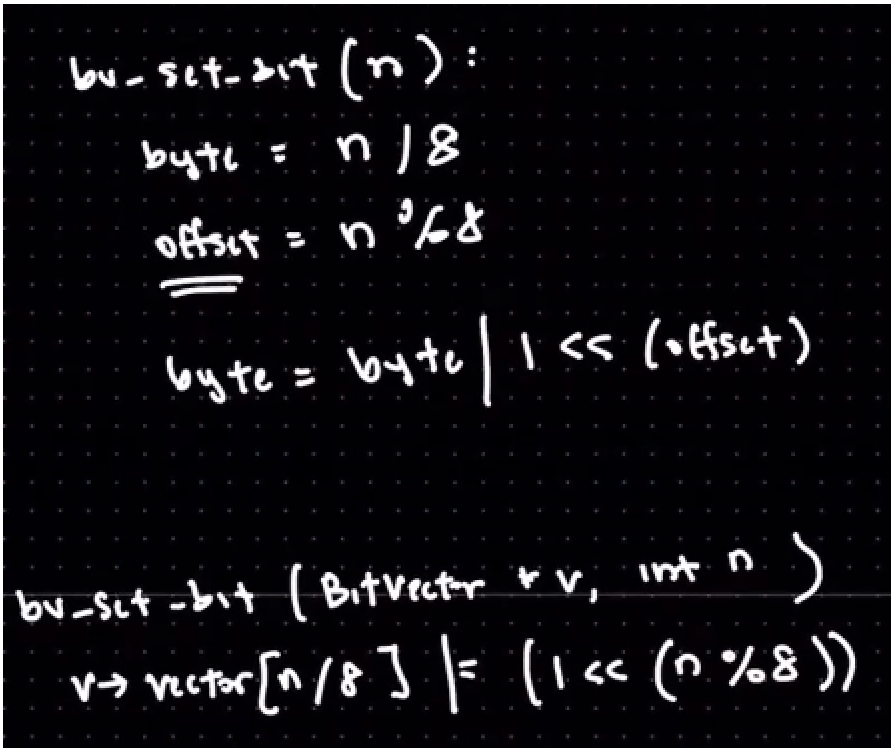
According to the lookup table there is no way to correct the code because the error code comes out to be 0 1 0 1 which is 1 0 10 in binary which corresponds to error code 10 in the table which is HAM_ERR.

bv.c (bit vector):

```
struct BitVector {
    uint32_t length; // Length in bits.
    uint8_t *vector; // Array of bytes.
};
```

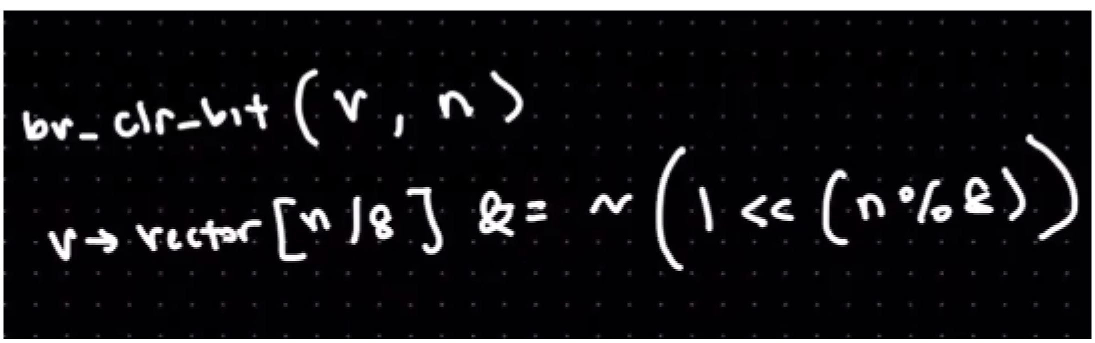
```
BitVector *bv_create(uint32_t length) {
        BitVector *bv = malloc( sizeof(BitVector) )
        bv-> length = length
        bv-> *vector = 0x0
        return bv
}
void bv_delete(BitVector **v) {
        if( *v )
                free( *v )
}

uint32_t bv_length(BitVector *v) {
        return v->length
}
```

void bv_set_bit(BitVector *v, uint32_t i) {



picture from Eugene's section

void bv_clr_bit(BitVector *v, uint32_t i)



picture from Eugene's section

what is happening here is you are taking a 1 and left shifting it by n%8 and then flipping all the bits, and then logical and'ing it with the bit vector of n/8

void bv_xor_bit(BitVector *v, uint32_t i, uint8_t bit);

similar to clr_bit, take bit and left shift it by n%8 and then xor it with the bit vector in vector[n/8]

uint8_t bv_get_bit(BitVector *v, uint32_t i){

uint8_t result = v-> vector[i/8]

return (result right shifted by (i%8))
}

bm.c (bit matrix): to be continued

```
typedef struct BitMatrix {
    uint32_t rows;
    uint32_t cols;
    BitVector *vector;
} BitMatrix;
```

uint32_t bm_rows(BitMatrix *m)
return rows

uint32_t bm_cols(BitMatrix *m);
return cols

void bm_set_bit(BitMatrix *m, uint32_t r, uint32_t c);
bv_set_bit(r * (columns in matrix) + c)

void bm_clr_bit(BitMatrix *m, uint32_t r, uint32_t c);
bv_clr_bit(r * (columns in matrix) +c)

uint8_t bm_get_bit(BitMatrix *m, uint32_t r, uint32_t c);
bv_get_bit(r * (columns in matrix) +c)

BitMatrix *bm_from_data(uint8_t byte, uint32_t length);
bm_create
for all the bits i in byte
        if bit i is 1
                bm_set_bit(i)

uint8_t bm_to_data(BitMatrix *m)
for i 0 to 7
        byte |= bv_get_bit(m) << i
return byte

BitMatrix *bm_multiply(BitMatrix *A, BitMatrix *B)

```c
Matrix *mat_multiply(Matrix *a, Matrix *b) {
  assert(a->cols == b->rows);
  Matrix *c = mat_create(a->rows, b->cols);
  for (int i = 0; i < a->rows; i++) {
    for (int j = 0; j < b->cols; j++) {
      int sum = 0;
      for (int k = 0; k < a->cols; k++) {
        sum += mat_get_cell(a, i, k) * mat_get_cell(b, k, j);
      }
      mat_set_cell(c, i, j, sum);
    }
  }
  return c;
}
```

picture from Prof. Long slides, code adapted from it

encode.c:
parse command line for options/input and output files
initialize generator matrix G by manually setting all the bits
while(fgetc(infile) != EOF)
    lower = lower nibble of fgetc
    upper = upper nibble of fgetc
    code1 = ham_encode(lower)
    code2 = ham_encode(upper)
    fputc(code1, outfile)
    fputc(code2, outfile)
close files/free memory

decode.c:
parse command line for options/input and output files
initialize transpose of parity checker matrix by manually setting all the bits
while (fgetc(infile) != EOF)
    fgetc again to get second byte
    call ham_decode twice for each byte passing in the address of a variable to store message
    switch(the value that ham_decode returns)
        HAM_ERR: uncorrected +=1, processed +=1
        HAM_OK: processed += 1
        HAM_CORRECT: corrected +=1 processed+=1
    repeat switch for second byte

```
        pack(msg1, msg2)
        fputc( packed byte )
if( verbose )
        print stats
close files/free memory

hamming.c
ham_encode(G, msg) {
        bitmatrix c = bm_multiply( bm_from_data(lower_nibble(msg)) , G)
        return bm_to_data(c)
}

ham_decode(Ht, code, *msg) {
        initialize lookup table
        bitmatrix asdf = bm_from_data(code)
        bitmatrix error = bm_multiply( asdf, Ht)
        if( lookup is ham_ok)
                *msg= lower nibble of code
                return ham ok
        if( lookup is ham_err)
                *msg unchanged
                return hamm_err
        else
                flip bit of code
                *msg = lower nibble of code
                return ham_correct
```