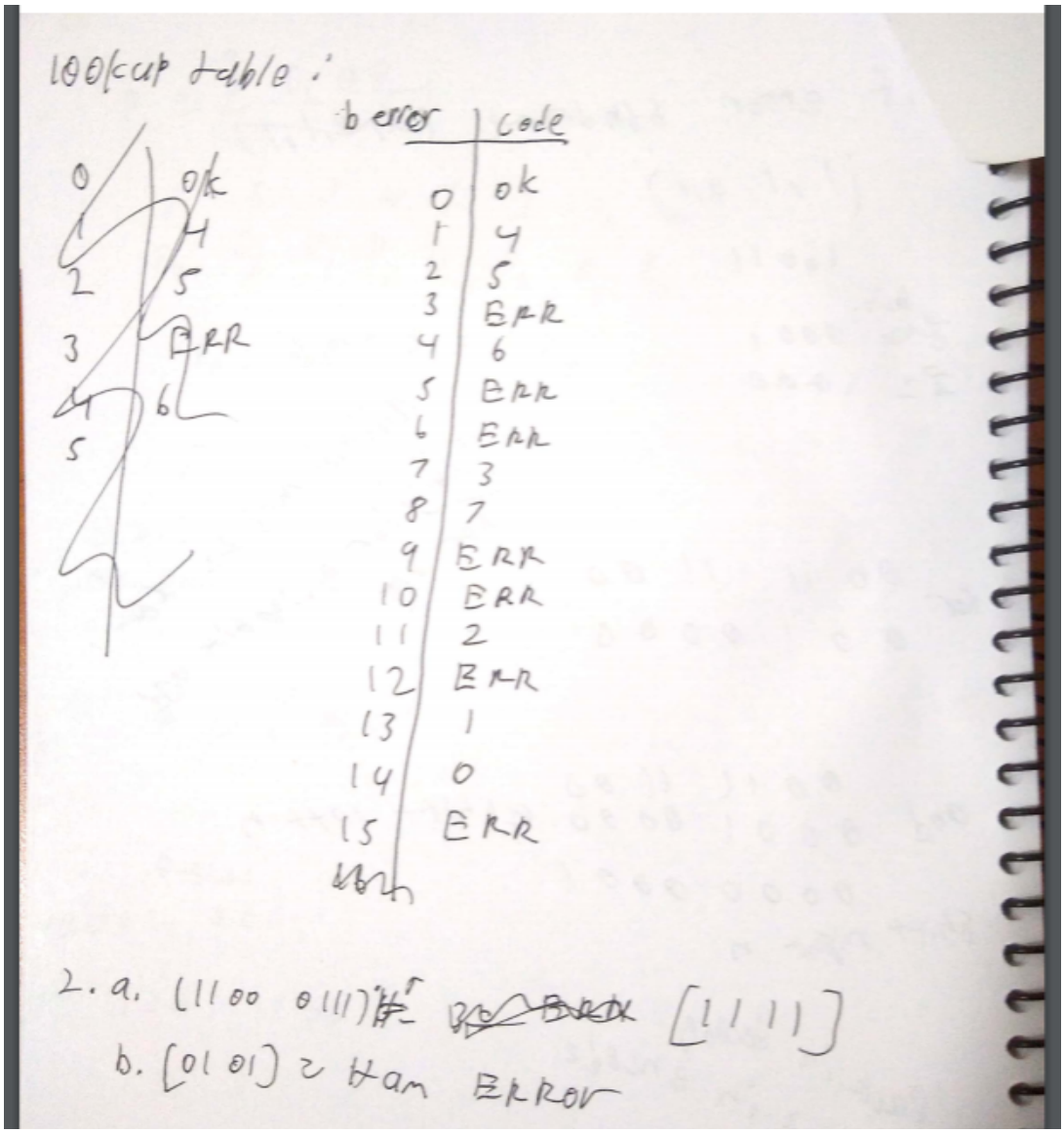
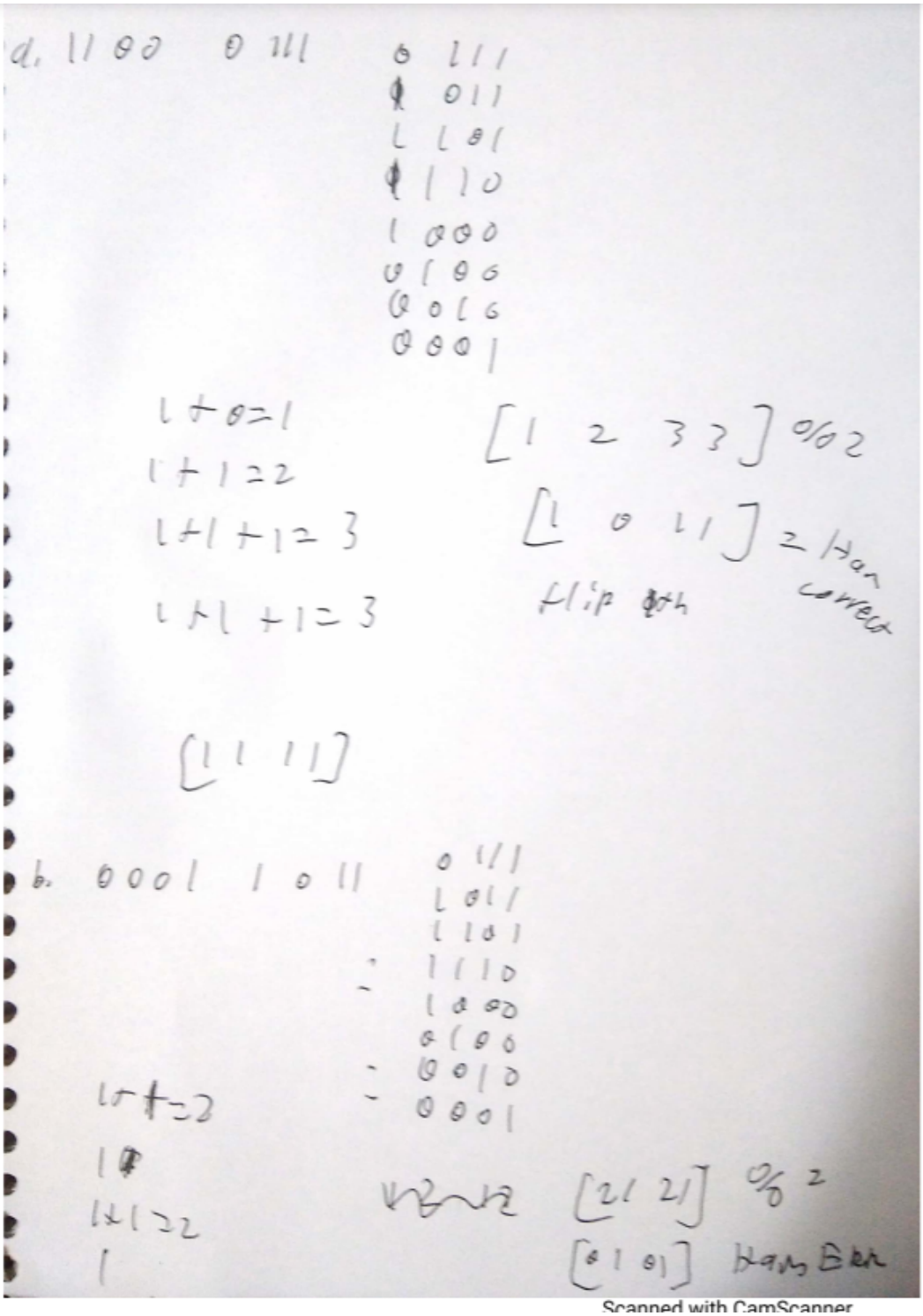
Winston Yi
wyi10@ucsc.edu
5/1/2021

CSE13s Spring 2021
Assignment 5: Hamming Codes
Design Document

PRELAB:





Scanned with CamScanner

OVERVIEW:

For this assignment, I have created a program that will encode a series of bytes into a series of 8:4 Hamming codes. I have also created a program that will decode the series of 8:4 Hamming codes into their original sequence of bytes. Additional programs are supplied that can inject noise into the encoded message and measure the entropy of a file.

TOP LEVEL DESIGN:

The top level design is shown by the following pseudocode:

```
overview of encode.c:
set command line options
go through file input and encode each byte
```

```
overview of decode.c:
set command line options
go through file input and decode each byte
```

```
overview of hamming.c:
set lookup table
ham_encode:
  turn byte into matrix, multiply by G
  set code to data
  return the data
ham_decode:
  take the code and multiply by Ht to get error syndrome
  find matching error syndrome in lookup table
  do ___ according to result of lookup
```

```
overview of bv.c: Basis to make bm.c
bv_create:
  creates a vector (OF AN ACCURATE SIZE)
bv_delete:
  deletes
bv_length:
  returns length
bv_xor_bit:
  xors a bit
bv_set_bit:
  sets a bit
bv_get_bit:
  gets a bit
bv_clr_bit:
  clears a bit
bv_print:
  prints out the vector IN VECTOR NOTATION NOT BINARY
```

```
overview of bm.c:
bm_create:
  creates a matrix
bm_delete:
  deletes
bm_rows:
  returns rows
bm_cols:
  returns cols
bm_from_data: BitMatrix from data TO MATRIX
  turns data --> vector --> matrix
bm_to_data: FROM BitMatrix TO UINT8_T
  turns a BitMatrix (which is really a vector) into data
 bm_xor_bit:
  xors a bit
bm_set_bit:
  sets a bit
bm_get_bit:
  gets a bit
bm_clr_bit:
  clears a bit
bm_print:
  prints out the matrix IN VECTOR NOTATION NOT BINARY
```

The BitMatrix ADT is supported by the BitVector ADT. The BitMatrix is simply understood as a matrix, but in reality is a BitVector that is abstracted to be able to perform the required matrix multiplication and conversion operations that are used later in the encoding and decoding in hamming.c, which is then used in the encode/decode files which encode/decodes the inputs.

DESIGN PROCESS:

As usual, I attempted to understand the math and reasoning behind the hamming codes. I made sure to learn matrix multiplication and review some basic linear algebra from AM10 that I had forgotten. I then created the lookup table and did a few examples of hamming codes by hand and checked their results with a tutor.

After, I created the basic ADTS, the BitVector ADT, and the BitMatrix ADT that would be essential to my programs. I thoroughly tested everything (except my BitMatrix ADT, which I just tested a few examples) and ended up not catching an error in the matrix multiplication function for my BitMatrix ADT which caused me much trouble later.

After creating my ADTS I then made my encode.c file. The command-line options were very similar to the previous assignment and posed little problem, as did the encoding function. However, the loop to get each byte and encode them had several issues, one being a false boolean expression I would always evaluate due to a missing parenthesis, another due to not properly taking into account integer division, and the final one being casting the wrong type to the bits I was reading in (a uint8_t instead of a character). The final issue was very simple but for some reason gave me immense trouble as well: how to use an integer as a uint8_t, when I never had to do anything in the first place. After getting the encode function to work the decode.c file was simple. The decoding function however was troublesome as I ran into segmentation faults while attempting to free my memory.
After getting my encode and decode files to work, I learned some basic Linux terminal commands such as “|”, “<</>>”, and “echo”. I then learned how to create bash scripts and execute them.

FINAL THOUGHTS:

Although I believe this assignment to be much work (again), I found this assignment the most satisfactory one because the code itself had a difficulty comparable to the previous assignment. The understanding of the logic and flow behind the math and ADT’s was the biggest part and I enjoyed that the most. In addition, I started my journey to learning how to use the terminal effectively which was much more fun than spending dozens of hours debugging small issues. As for my code itself, it was rather slow when running which is understandable given that it runs over 200000 bytes every time it encodes or decodes, not even including the error and entropy. I also understood the process behind improving runtime by a constant as even decreasing the runtime of my bash script by a factor of 8 would have allowed me to run it much faster than the hour it took.

Understanding each component was also rather difficult and it was much easier to debug the code that I thought up of and wrote myself than the code I wrote while following the TA pseudo code provided to us. Knowing what the BitMatrix was implemented as and also knowing the structure we were told to understand as and how they could be converted from one to the other helped me write my ADTs. Also, implementing the code myself was faster and easier to debug than pseudocode given by the TA’s. For example, writing the matrix multiplication function was much faster and easier than attempting to debug the code based on the TA pseudo code.