Assignment 6
Design Document (Draft)

I am still in the early stages of assignment 6, so naturally, my strategy is not entirely flushed out yet. Below are my design materials so far.

Purpose

Create a two programs: one that can compress a given file using Huffman coding, and one that can decompress a file.

Pseudocode to illustrate layout/structure and to describe/explain each part of the program

```
I just wanna understand how encode is actually gonna work.
i want to be able to visualize how this is supposed to play out before i go about coding it


super basic overall idea:
// get data from infile
// encode and compress data
// write data to outfile

slightly more specific:
// read an input file
// find Huffman encoding for file
// use Huffman encoding to compress file and write compressed version to outfile


more specific set of instructions:
// read input file
// step through each character and compute histogram of file (count number of occurrences of each character)
// construct Huffman tree using histogram and priority queue
// construct a code table. you will need stack of bits to traverse Huffman tree
// traverse tree and dump an encoding of the Huffman tree to a file
// step through each character of the file again. this time, emit each character code to outfile


refined pseudocode:
Create Histogram
     Initialize array of 256 uint64_t values all set to zero
     Read through infile and increment values of histogram accordingly
     Increment the count of element 0 and element 255 by 1

Create Priority Queue
     For each symbol in histogram create node and insert it into priority queue.
     (the symbol and frequency is given by histogram, but what should the left and right node be? i suppose they should be nothing)
     (also, do we construct the priority queue within build_tree(), or before calling it? seems like it should be done within)
     UPDATE: it should be within build tree

Construct Huffman Tree from Priority Queue using build_tree()
     while there are two or more nodes in the priority queue …
          dequeue 2 nodes
          the first will be left child node
          the second will be right child node
          join these two nodes together using node_join and add joined node back to priority queue
          (the frequency of the parent node should be the sum of the frequencies of the left and right child nodes)
     eventually, there will only be one node in the queue. this is the root
     return the root node of the tree you just built
     (wait, so the original priority queue we created gets transformed into the Huffman Tree? Seems like it)
     I guess when you return the root you are effectively returning the entire tree,
     because the root contains the addresses of all the locations of data in memory

use build_codes() to construct code table by traversing Huffman Tree. code table is an array of 256 Codes (Code is a stack of bits)
     initialize a new code c using code_init()
     starting at the root of Huffman Tree, perform post-order traversal (recursion?)
          if current node is a leaf ... (base case?)
               current code represents path to this node
               current code is code for this node's symbol
               so add it to code table
               return and go back up
          otherwise current node must be an interior node (recursive case?)
               so push 0 to code and recurse down left link
```

```
  // read compressed file
  // decompress file
  // write decompressed file

  // read the emitted (dumped) tree from input file. reconstruct the tree using a stack of nodes
  // read the rest of the infile bit-by-bit, traverse down Huffman tree, and write characters to the outfile
```