13. Huffman and Canonical codes with examples.

Alphabet:

- a-z
- ! ? . , ; :

Note: The processor is 64-bit

The lookup table used depends on the version of Huffman Encoding we use.

Struct for tree nodes (regardless of encoding method):

```
struct Node {
   char symbol;
   int frequency;
   struct Node* left;
   struct Node* right;
};
```

Struct for table entries:

The "code" array is of length 31 because our alphabet is composed of 32 symbols, meaning 31 is our maximum depth

```
struct tableEntry{
   char symbol;
   char code[31]
};

struct tableEntry[32]; //32-sized alphabet.
```

For assigning codes

- *We build the min-heap like usual*
- *We assign the most frequently occurring symbol 0 (initialization)*
- *For each symbol after, we*
    - *Check if its depth is greater than the previously visited symbol*
        - *If so, we << 1*
        - *If not, do nothing*
    - *Assign the current code*

    - *Then add 1*

Example for above:
A = 2
B = 3
C = 3
D = 4

A = 00
Current code = 01 (after increment)
B = 010
Current code = 011
C = 011
Current code = 100
D = 1000

This allows the program to only have to transmit the lookup table as opposed to the tree, since the codes associated with each symbol tell us the length (depth) they have in the tree.