Simple UML class diagrams

HuffmanCompress
- Priority Queue of HuffmanTrees - Frequency Counter table - BitStream object to write and read from files - HuffmanCode object to determine sequence of 0's and 1's
- Contains main() - Handles the command line arguments and issue errors if incorrect input - Reads and writes to and from files

BitStream
- sizeHuffman → indicates number of bytes the Huffman tree representation takes up in the file - binarySequence → represents the binary sequence of 0's and 1's to traverse the tree - Frequency counter object → represents the frequency table
- Loads data into private members - Write data to output stream, given the name of the file - Reads data from the input stream, given the name of the file

FrequencyCounter
- Count[256] array → integer array to keep count of the frequency of each character in a file
- Given the string to a file, it counts the frequency of each character in the file

HuffmanCode
- Code[256] array → string array to keep the binary sequence of each letter to traverse the tree
- Given the Huffman tree root, it creates the binary sequence for every character that exists in the tree

HuffmanTree
- Weight → the total weight of the tree - root → BinaryNode pointer to the root
- creates a binary tree that has all the characters at the leaves of its tree - able to compare trees by their weights through an overload operator

Format of compressed file

<table>
<tr><td>Total # of bytes written in this file representing the Frequency table</td><td colspan="2">Frequency table that consists of the character followed by the integer representation of its frequency.

Ex) c5F2s6&1A1 …</td></tr>
<tr><td colspan="2" rowspan="2">Binary string that represents each character in the correct order to traverse the Huffman tree to extract the message</td><td></td></tr>
<tr><td>End of File Marker</td></tr>
</table>

Idea behind Compression:

- Get the frequency of each character in the file we want to compress
- Store the frequency table such that the character would appear in the file followed by its frequency
- Make the Huffman tree with the frequency table
- Find the Huffman code of each character in the Huffman tree
- Create a binary string of each character in the file in the correct order to make a large long string of 0's and 1's and store this in the file

Idea behind decompression:

- Extract the # of bytes written in the file that represents the frequency table, read the frequency table by noting the character and mapping it with the corresponding frequency (which is followed by the character in the .huff file)
- Recreate the Huffman's tree using the frequency table we created
- Extract the binary string that represents each character in the correct order to traverse the Huffman tree to get the message
    - When reaching a leaf, add character to final string that will be written to the file