The core of my approach to solve the problem is to use a structure for the Huffman coding. This structure holds the number of occurrences of a letter and the pointers to the nodes children. If it is a leave node it will also contain the actual letter value. Using this technique, it can be combined with a wide variety of helper programs to break the problem down into smaller parts. I have helper programs that build new structure nodes, check if a node is a leaf node, free nodes, compare nodes, and free the whole tree. The task can also be broken down in to several main functions. One main function will initially create an intermediate linked list to be referenced by the Huffman coding programs. One other main function then creates the Huffman tree using that linked list and calling its helper functions while using the tree structure I have created. This code can then return the address of the main head of the whole tree. Using this address the tree can be sent into another file.

We know that we can only write to a file one byte at a time, however we are writing a single bit in the form of a 0 or 1 for the path in this Huffman tree that has been created. What can be done is that once the path is figured out a helper program can be called that will write the one bit it receives as input to an output file. However, this program can check if all bits it has received constitute a whole byte yet. If they do not it can keep track of how many with a counter. Once this counter reaches 7 (for 0 based bytes) we have a full byte. So, we can then write this whole byte to the output file. Once the Huffman coding has finished if there is leftover room EOF characters can be used and padded at the end of the byte if when done the program does not have a whole byte to send over. Using this method, the code will periodically write to the output file only when a full byte is created. Otherwise it will not be actively writing to the output file. Similarly, when un-coding we can read these bits in one whole byte at a time. We load them into an array that we can use to reference while decoding the Huffman tree . Once we are at the end of the array we can reuse that space to store the next byte. This will help preserve some of the space complexity of the Huffman coding. Once an EOF character is reached we know that the Huffman tree does not need anything more read from it and we can stop reading in bytes of information.

When decoding the Huffman tree we can use the technique mentioned above to get arrays that have the path in the provided Huffman tree. We simply follow this path and at each level we move to a left or right (0 or 1) child we check if it is a leaf node. If the node is a leaf node we know that it will contain the character that we are looking for and we can output that character to decoded file. We can repeat this process until we get an EOF character and know that the whole phrase has been decoded. After that helper functions will free the Huffman tree and all arrays and lists used to create the new output files.