First, I will write a huff.h for function declarations, I'll also create a structure called Node that contains a char “ch” and a int “wt” and 2 pointers to Node called “left” and “right”, and another one called Nodehead that contains a pointer to a Node.

For huff.c, I'll write several different functions in it to do the Huffman encoding efficiently.

main: check whether there is an argument use argv, then read the file content into a string called input use argc. Then allocate memory and initialize an array of Node called “elem” with a length same as the input string, then call function freq with both input string and “elem” array as parameters. Deep copy elem to elem2 to save the array. Then call Build with “elem” and an initialized Nodehead “head”. Then call write with the elem2 and input filename. Then call Freeall with the head to free the tree, Nodehead and elem2.

freq: use a loop to read through every element of the input string. And in each iteration, read a character from input, and if it is not contained in any of “ch” in the current “elem” array, save it to an empty “ch” the “elem” array and put a 1 at corresponding “wt”. If the read-in character is already in one of the “ch” in the “elem” array, just add 1 to the corresponding “wt”. After finished the loop, sort the “elem” based on the numbers in “wt” in descending order.

Build: use the method given in the instruction to build a binary tree with the root pointed by the head, remember to allocate memory for each internal Node and add up the “wt” of its branches and store it in its “wt”.

Write: write down the array in first line, elements separated with semi cone, character and weight separated by comma, then write EOF character at the start of second line and generate the sequence of bits and write them in second line and end up with the EOF character. Read the length of the filename and add “.huff” at the end of it to create output filename

For unhuff.c:

Main: check whether there is an argument use argv. Then read through the first line of file to generate the elem array and call Build again to get the tree. Then read first character of the second line to find the EOF character, then read the bits till the EOF character shows again. Then un bits the sequence and write the output down in a string based on the rebuild tree. Then write them into a file.