Assignment 5: Huffman Coding
This program encodes a user inputted file via Huffman encoding, compressing it to reduce the amount of storage spa ce for said file.
The program can then decode either previously encoded file or any other Huffman encoded file to provide an exact r eplica of the original
uncompressed file.

node.c:
File that stores the Node struct and ADT that allows for the storage of symbols and their respective frequency in the node object.
The Node struct has four variables inside
-64 bit integer frequency
-8 bit integer symbol
-2 node pointers, allowing for it to point to the node to the left and right of it
The Node also allows for the joining of two nodes via a parent node where it
-creates a third node separate from the two inputted nodes
-sets the third node's left and right pointers to the two inputted nodes

pq.c:
File that stores the PriorityQueue struct and ADT that allows for the storage of nodes in order from lowest frequency to highest frequency.
Works almost identically to a normal queue with the exception that newly added nodes need to be placed in order to conform to the proper ordering.
Long recommends implementing priority queue as a min heap instead of an insertion sort, I will likely implement it as an insertion sort at least to start.
The implementation of insertion sort is rather simple
-Starting at the end of the array scan to the front one element at a time
-If the current element scanned is less than the to-be-inserted element then insert said element to the right
-Else continue scanning til the front of the array, if no element is found to be smaller then insert the element at the v ery front

code.c:
File that stores the Code struct that allows for the storage of an array of bits, used when reading in individual values of characters.
Fairly straightforward, the struct has
-32 bit integer top
-8 bit integer array bits of size MAX_CODE_SIZE
All of the functions simply enable changing the value at specific points in the array or return basic values regarding t he Code object.

io.c:
File that does the actual reading of the user inputted file to then store those values for later use.
-can read the bytes if the infile
-write bytes to the outfile
-read a specific bit from the infile by reading a byte and read the buffered bits one at a time
-store the bits into the Code ADT
-kill any leftover bits after output file has been encoded/decoded

stack.c:
File that stores the Stack struct and ADT for storing nodes in a standard stack.
Makes use of an array of pointers to nodes to store said nodes.
-Relies on an integer to keep track of the next available index in the array
-Can return all of the necessary values that a standard stack can return, popped value, pushed success, size, boolean empty

huffman.c:
File that stores the logic needed for the Huffman tree logic.
A Huffman tree is similar in nature to a binary tree with the key exception being that each leaf on the tree correspond
s to a letter and
its frequency in the provided file.
 -Can construct a huffman tree from a provided input preencoded file
  -returns the root node of the constructed tree
  -uses static arrays in lieu of standard arrays
 -Build a Code from a preprovided Huffman tree
  -constructed codes are copied to the code table, a precreated array of codes
 -Do a post-order traversal of a Huffman tree and writing the results to the outfile
  -writes an 'L' before the symbol of every leaf
  -writes an 'I' before interior nodes
 -Rebuild a Huffman tree based on the post-order tree dump found in the tree_dump array

encode.c:
Combines all of the other files together and acts as a main file.
Requests user input with getopt
 -h: Prints out a help message describing the purpose of the program and the command-line options it accepts, exitin
g the program afterwards.
 -i infile: Specifies the input file to encode using Huffman coding. The default input should be set as stdin.
 -o outfile: Specifies the output file to write the compressed input to. The default output should be set as stdout.
 -v: Prints compression statistics to stderr. These statistics include the uncompressed file size, the compressed file siz
e, and space saving.
The program then begins the process of encoding the user inputted file
 -Counts the number of occurrences of each unique symbol within the file, creates a histogram
 -Contructs a Huffman tree based on the histogram through the use of a priority queue
 -Builds a code table with each index representing a specific symbol and and the corresponding frequency
 -Prints an encoding of the Huffman tree to an output file through the use of a post-order traversal of the Huffman tre
e
 -Repeating the cycle for each symbol in the input file

decode.c:
Combines all of the other files together and acts as a second main file.
Requests user input with getopt
 -h: Prints out a help message describing the purpose of the program and the command-line options it accepts, exitin
g the program afterwards.
 -i infile: Specifies the input file to decode using Huffman coding. The default input should be set as stdin.
 -o outfile: Specifies the output file to write the decompressed input to. The default output should be set as stdout.
 -v: Prints decompression statistics to stderr. These statistics include the compressed file size, the decompressed file
size, and space saving.
The program then begins the process of decoding the user inputted file
 -Reads the dumped tree from the input file, needs to read a stack of nodes in order to properly rebuild the tree, fails i
f it cannot read the stack
 -Reads the input file one bit at a time proceeding to slowly recreate the Huffman tree one node at a time
  -if a 0 is read then it walks down the left node
  -if a 1 is read then it walks down the right node