Assignment 5- Huffman Coding

Description:

This program will encode a script given to it using the Huffman code. It will utilize stacks to build the Huffman code while parsing through the binary tree. It will shrink the size of bytes required to transport the file and inorder to read the file once more you will be required to use the Huffman code to remake the tree as print out the script by decoding it. We are able to take the file as an input and output it to a current file or create a new file.

encode.c: This file will contain your implementation of the Huffman encoder.
decode.c: This file will contain your implementation of the Huffman decoder.
defines.h: This file will contain the macro definitions used throughout the assignment.
header.h: This will contain the struct definition for a file header.
node.h: This file will contain the node ADT interface.
node.c: This file will contain your implementation of the node ADT.
pq.h: This file will contain the priority queue ADT interface.
pq.c: This file will contain your implementation of the priority queue ADT. You must define your priority queue struct in this file.
code.h: This file will contain the code ADT interface.
code.c: This file will contain your implementation of the code ADT.
io.h: This file will contain the I/O module interface.
io.c: This file will contain your implementation of the I/O module.
stack.h: This file will contain the stack ADT interface.
stack.c: This file will contain your implementation of the stack ADT. You must define your stack struct in this file.
huffman.h: This file will contain the Huffman coding module interface.
huffman.c: This file will contain your implementation of the Huffman coding module interface.
Makefile: This is a file that will allow the grader to type make to compile your programs.
• CC = clang must be specified.
• CFLAGS = -Wall -Wextra -Werror -Wpedantic must be included.
• make should build the encoder and the decoder, as should make all.
• make encode should build just the encoder.
• make decode should build just the decoder.
• make clean must remove all files that are compiler generated.
• make format should format all your source code, including the header files.

README.md: This must be in Markdown. This must describe how to build and run your program.
DESIGN.pdf: This must be a PDF. The design document should answer the pre-lab questions, describe the purpose of your program, and communicate its overall design with enough detail such that a sufficiently knowledgeable programmer would be able to replicate your implementation.

encode.c pseudocode:

Utilizes multiple commands through arguments and get opt

-i Reads input file provided following the implemented

-o Writes output file to the specified file

-v prints out compression stats such as size,and space saving

While loop through input file and tallies occurrence of each unique symbol

Using the stored occurrences ultized priority queue to build a Huffman tree

Node.c pseudocode:

node_create(symbol,frequency)

Create node by allocating memory using malloc and taking parameters of symbol and frequency

Sets symbol and frequency to the Node values and returns the node

node_delete()

Frees the left and right nodes and main node

Node_join takes left and right nodes and returns a parent note that has the combined frequencies of the nodes and has left and right as part of the parent node properties.

node_print(prints of nodes symbol and frequency)

Pq.c pseudo code:

Priority Queue struct has capacity,top and array of Nodes items and creates memory for it.

Pq_delete frees items and q

Pq empty returns true if top equal to 0

Pq_full returns true if full

Pq_size returns size of queue

Queue sort the queue in order or frequencies from low to high

Enqueue adds Node to queue and sorts the queue and returns if it was successfully executed

Dequeue pops off the 0 index node and passes through the parameter n.

pq_print(prints the queue)

Io.c pseudocode:

read_bytes()

Takes in a Block(4096) of bytes and reads it from the infile and passes the bytes into the buffer

write_bytes(Takes in a buffer of bytes and writes it to the outfile)

read_bit ()

Uses a static buffer to store bytes and through the bytes is able to parse through it using bitwise operators to get each bit