Assignment 7
Lempel-Ziv Compression

Program Design

There are two programs in this assignment, encode and decode. The main functions for each program are in their respective .c files. The encode program encodes performs LZ78 compression on any text or binary files and decode program performs decompression.

Encode.c: Follows the pseudocode given on assignment pdf.

Decode.c: follows the pseudocode given on assignment pdf.

Io.c: Inspired from guidance given from Oran. Deals with the input of files being read and outputs the encoded or decoded file depending on the program you run.

Word.c: Build a word table and words for the decode program. The word table is used to store symbols and their codes. Also contains helper functions for the word table and word object.

Trie.c: Builds a trie object which is used in encode program. Reads file and stores its symbol and code in trie connected with to its previous symbol trie node. Contains helper function for the trie as well.

Program Pseudocode

Io.c:

```
read_header():
        Read bytes from file. Call read_bytes

write_header():
        Write bytes to the outfile. Call write_bytes

read_bytes():
        Bytes_to_read = to_read
        Bytes_read = 0
        Total_read = 0
        Do:
                read() and store in bytes_read
                Decrease the bytes you have to read by bytes you read in the previous line
                Increase the total read by the number you read
        While there are bytes to read and total_read != to_read

buffer_pair():
```

```
        For i->bit_len:

                Buffer bit starting from lsb
                Increment bit_count
                Check if bit_count reaches the end of bit_buf:
                        If does then write the bytes to outfile and reset it

        For i -> 8:
                Buffer index starting from lsb
                Increment bit_count
                Check if bit_count reaches the end of bit_buf:
                        If does then write bytes to outfile and reset it

read_pair():
        *code = 0
        *sym = 0
        For i->bit_len:
                If bit_count == 0:
                        read_bytes()
                Buffer bit starting from lsb
                Increment bit_count
        For i->8:
                If bit_count == 0:
                        read_bytes()
                Buffer index starting from LSB
                Increment bit_count
        Return *code!=STOP_CODE

buffer_word():
        Increment the total number of sym var with word length
        For i->w.len:
                Sym_buf[sym_count] = w->sym[i]
                Increment sym_count
                If sym_count reaches end of sym_buf:
                        write_bytes() to outfile
                        Reset sym_count

flush_words():
        If sym_count > 0:
                Write_bytes to outfile
                Reset sym_count

flush_pairs():
        If bit_count > 0:
```

Decrease bit_count to a byte value
write_bytes() to outfile
Reset bit_count

bit_len():
//gives length of bits of a value passed in

Trie.c:

trie_node_create():
Calloc space for object trie node

trie_node_delete(node):
If node is not null then free it

trie_create():
Calloc space for 1 trie_node...this will be the first node.. The root
Set the root's value to empty code

trie_reset():
Loop through the trie and delete it starting START_CODE node

trie_delete():
Loop through the trie and delete it

trie_step():
if the node of that symbol is not null then return it

Word.c

word_create():
Allocate space for word
Calloc space word-sym //the symbols
Initialize the data in word->sym to what was passed in sym

word_append_sym():
Check for empty word
If word is empty then create a new word with that sym and return
Else create a word for all symbols in the word with the new symbol

wt_create():
Calloc space of MAX_CODE
Initialze the first word to be empty word

wt_delete():

Loop through the word table

Delete every word that isn't null

Free word table

Encode.c and decode.c follow the assignment's pseudocode