PSEUDOCODE

encode.c

Main:
Take in command line arguments if needed

while(Getopt (“ v , i ,o “) not EOF):
    Case v: print stats
    Case i: include input file
    Case o: include output file

open(file to read, stdin if not specified), read only;
open(file to write, stdin if not specified,write only;
Write in a FileHeader with protection(sizeof32)

Initialize trienode at root to be NULL;
While there is symbol in file:
    Create a trienode;
    Trie_step to check if there’s childrens
    If trie node has children:
        Prev_node = current node
        Curr_node = nextnode
    Else:
        buffer _pair (stores binary in array)
        Create a children for trienode and assign with a code
        Curr_node = root:
        Increment code
    If code == MAXCODE:
        Reset trie node back to root
if(current node is not the root):
    Buffer_pair until root;
    Increment code
Buffer remaining pair
Flush out your buffer

Print Stats if -v flag specified

Trie.c

Trie_node_create(code);
- Malloc size for a trie node
- Set each index of the children to null
- Assign struct code with the given code
- Return node

trie_node_delete(t):
- Free node;

Trie_reset:
- Delete and free all children of each trie node
- Set the root back to code of 1

Trie_create();
- Calls trie_node create
- Set the node to code to 1
- Return the creation

Trie_delete(t):
- Delete the whole trie and its childrens

Trie_step(t):
- Checks if a children exist
- Return the children node

DECODE.C

Main:

getopt(“v,i,o”) until EOF;
Case v: print compression stats
Case i: input a input file
Case o: input a output file

Read infile(stdin if not specified) for read only or create
Read outfile(stdin if not specified) for write only or create

Read in file header and protection;

Create a WordTable

While there is a pair of symbols and code to be read from:
    Table[next code]= word_append_sym(table[curr_code],symbol);
    Buffer your code
    Increment next code

    If table reaches max code:
        Reset table
Flush_words out;
If -v flag then print stats;

Word.c

Word_create(symbol,len):
     Create a word struct and malloc size
    Word->syms = calloc size of symbol
    Copy symbol to the newly made word->syms
    word->len = len
    Return word

Word_append(*w,sym):
    If (w is NULL):
        word_create(symbol ,length of symbol)
        Append the symbol to the end of Word
    Else:
        Word_create (*w->syms , w->length+1);
         Append symbol to end of Word
        Return word

Wt_create():
    Create a table with MAX_CODE index:
    Set word table at index 0 to NULL and length of zero
    Return word table
Wt_reset();
    wt_delete();
    Reset word table to only index at 0
Wt_delete();
    Set all index to NULL

IO.c

Create two static buffers for encode and decode each of unit8_t;
Create a static variable called end to keep track of file number

Create a static bit_index that keep tracks of bit position

Read_Header(*Header):
    Read in sizeof(FileHeader)
    If whatever is read in does not equal to the magic number 0x8badbeef:
        exit(1);

Write_header(*header):
    Assign header->magic with 0x8badbeef
    Create a protection stat to header
    Write header bytes into outfile

read_sym(infile,sym):
    If index reaches 4096 or first time reading:
        Read in a block of bytes (4096 characters)
        Store in a static variable array;
        Create static index=0;
        If(nothing to be read);
            Return false

    If index reaches end of file:
        Return false

    If a symbol exists in array at the index:
        Return the character;

    Else:
        Return false;

Void buffer pair(outfile,code,symbol,bitlen):
    for(i in range bitlen):
        Store bit(1 or 0) of code in a buffer[bit_index/8];
        Bit_index++;

    for(i in range 8):
        Store bit(1 or 0) of symbol in a buffer[bit/index/8]
        Bit_index++;

    If array is full at any point:
        Then Flush pairs(outfile);

```
Flush_pairs();
        write(outfile,buffer,buffer size)
        bit_index=0;


Bool read pair(infile,*code,*sym,bitlen):
        If (bit_index is 4KB or first time reading);
                Read in size 4KB to buffer from infile;
                if(no bytes read):
                        Return false;

        For i in range(bitlen):
                if(buffer[bit_index]==1:
                        Store 1 to temp buffer at i;
                Else:
                        Store a zero at temp buffer
*code =temp buffer


        For i in range (bitlen):
                if(buffer[bit_index]==1):
                        Store 1 to temp buffer
                Else:
                        Store 0 to temp buffer at i;
*symbol = temp buffer

If code is EMPTY_CODE and symbol is NULL:
        Return false
If at any point, temp buffer is filled:
        flush_word();
Return true


Buffer_word(w);
        buffer[index]=w->syms
        If buffer filled:
                Flush_word

Flush_word(outfile);
        write(buffer);
```